MASTER IN COMPUTER SCIENCE

Interactive visualization of large data sets with the Java and the virtual reality modelling language technology

El Ansari, Abdesamad; Vanbrabant, Alain

Award date:
2001

Facultés Universitaires Notre-Dame de la Paix,
Institut d'informatique
Namur

Academic Year 2000-2001

Interactive Visualization of Large Data Sets with the Java and the Virtual Reality Modelling Language Technology

Abdesamad El ansari
Alain Vanbrabant

Thesis submitted in fulfilment of the requirement
for the degree of master in Computer Science

Facultés Universitaires Notre-Dame de la Paix,
Institut d'informatique
Namur

Année Académique 2000-2001

Visualisation de grands ensembles de données via les technologies JAVA et VRML

Abdesamad El ansari
Alain Vanbrabant

Mémoire présenté en vue de l'obtention
Du grade de Maître en Informatique

Abstract

The purpose of the proposed thesis is validation of Wesson's arguments that Java technology is mature enough to provide the tools for data visualization as defined in the **Introduction**. The objective will be to show that applications can be rapidly mounted that will allow an end user to view the data in a number of different ways, interactively and in real time.

For this thesis we will focus on the Interactive Visualization of large data sets and show, by means of a case study, how this can be implemented using chiefly the Java and VRML technologies. The main result will be the visualization of three-dimensional charts in virtual worlds, and the architecture of the implementation will be object-oriented, in order to facilitate easier extension of the application. Finally, so as to allow a user to create dynamically such virtual charts, we have to provide him with a usable and useful User Interface.

In the light of the above, a critical assessment will be made of the current (and possibly future) ability of Java to deliver such applications, and of the Virtual Reality to allow such an Interactive Information Visualization.

Keywords :

Visualization, Interactivity, Three-dimensional Representations, Virtual Reality and Virtual Environments, Charts, Data, User Interface, Java, VRML, extension and reusability.

Résumé

Le but de ce mémoire est la validation des arguments du Professeur J. Wesson, qui affirme que Java est une technologie assez mûre pour fournir des outils à des fins de visualisation de données, comme défini dans l'**Introduction**. L'objectif est de montrer que des applications peuvent être rapidement mises sur pied, et permettre à un utilisateur final d'analyser des données sous différentes formes, de manière interactive et en temps réel.

Ici, nous allons nous concentrer sur la Visualisation Interactive de grands ensembles de données et montrer par une étude de cas, comment ceci peut être implémenté à l'aide des technologies Java et VRML. Le résultat principal en sera la visualisation de graphiques 3D à l'intérieur de mondes virtuels, et l'architecture général de l'implémentation se devra d'être orientée objet, afin de faciliter toute possible extension de l'application. Enfin, pour permettre à l'utilisateur de générer dynamiquement de tels graphiques virtuels, nous nous devons de lui fournir une interface utile et utilisable.

A la lumière de ceci, une évaluation critique sera effectuée et concernera les possibilités de Java à produire de telles applications, et celles de la réalité virtuelle à permettre une visualisation interactive d'informations.

Mots-clés :

Visualisation, Interactivité, Représentations 3D, Réalité Virtuelle et Environnements Virtuels, Graphiques, Données, Interface, Java, VRML, extension et ré-utilisabilité.

Acknowledgments (Remerciements)

Nos remerciements les plus sincères s'adressent bien entendu à nos parents et à notre famille respectives, mais aussi à tous nos amis. Ces cinq années n'ont pas toujours été faciles, souvent pour nous, et parfois pour eux. Alors, merci pour leur précieux soutien et leur indispensable présence.

Au Professeur François Bodart, pour son suivi rigoureux, de l'ébauche à la clôture du mémoire. Nous voudrions particulièrement le remercier pour les nombreux conseils et critiques dont il nous a gratifiés.

To Professor Janet Wesson, from the University of Port Elizabeth, for her guidance and advice during our training at UPE. Also for the time she gave us during almost five months. We would like to thank her, as well as her family and friends, for their warm welcome in this so wonderful country, South Africa.

To the whole staff of UPE, professors and secretaries, for their kindness and warm receptions. And of course to all the people who allow us to have a so nice stay in Port Elizabeth.

A l'équipe Vésale, Christophe Grosjean et Christelle Darville, qui étaient très souvent là, dans leur bureau, pour nous conseiller et nous aider à résoudre certains problèmes.

A Vincent Lagneau, pour l'impression colorée de ce mémoire.

Finalement, à toute l'Université de Namur, qui nous a accueilli et supporté pendant cinq années, et qui a accepté de se détacher de nous durant cinq mois, afin que nous puissions effectuer notre stage en Afrique du Sud.

Contents

Contents :

Table of Figures :

Introduction

Introduction

I. *Problem definition*

Our project concerns **data visualization**, and more principally the interactive visualization of large data sets using the *Virtual Reality*[1] technology.

Due to the large amount of information available on the World Wide Web, there is a huge need to be able to visualize this information in ones own way, interactively and in real time. Although many tools exist for data visualization, these tools are embedded in specific packages and are not accessible to a Web author. In addition, most of them do not allow real interactivity.

Wesson has argued that Java and especially JavaBeans can change this lack of **non-accessibility** [Wessona2000]. Java's reusable software component model is referred to as *JavaBeans* [Deitel&1999]. JavaBeans (often called *Beans*) allow software developers to reap the benefit of rapid application development in Java by assembling predefined software components to create powerful applications and applets[2]. Graphical programming and design environments that support Beans provide programmers with tremendous flexibility by allowing programmers to **reuse and integrate** existing components. Others can link these components together to create applets, applications or even new Beans for reuse [Tebbut1999].

The Java Gauges (an IBM's family of Java's components) is based on this idea of Beans [Tebbut1999]. Gauge Beans are changing the face of data by providing the ability to create a range of measurements and scales that illustrate and animate information, including LED counters, needles, oscilloscopes, and bar indicators. No more static graphs or charts. No more updating stale numbers. No more old news. Gauge Beans connect to data providers and work with current values, which means data is fresh, up-to-the-minute, and more importantly, informative. The result is a variety of visual devices that make presenting information more than just presentable: progress is displayed in numerical fashion, with LED counters or rolling counters; growth can be illustrated through needles or RAG needles; developments can be depicted by thermometer, oscilloscope, or trace recorder; and, advances can be made plain with light bulb, block, or bar indicators. (See Figure 1)

---

[1] The concept of *Virtual Reality* (and of *Virtual Environments*) will be deeply discussed in the chapter two related to the Assessment of Interactive Visualization.

[2] An application is a program such as a word processor program, a spreadsheet program, a drawing program, an email program, etc. that is normally stored and executed form the user's local computer. An applet is a small program that is normally stored on a remote computer that users connect to via a World Wide Web browser. Applets are loaded from a remote computer into the browser, executed in the browser and discarded when execution completes. [Deitel&1999]

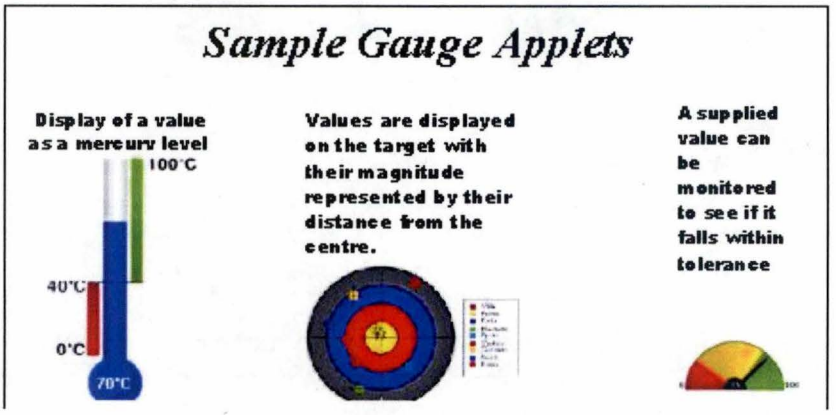

Figure 1: Examples of IBM Gauge Applets

But after some research, and with Wesson's agreement, we have concluded that Java Beans are not suitable for fairly large applications like the *Visualization Application*[3] we have focused on. Java Beans are more appropriate to small independent programs which can then be reused by other developers.

As we have presented it before, Java Beans is a component architecture for Java. It is a set of rules for writing highly reusable software elements that can be linked together in a "**plug and play**" fashion to build applications. Java Beans are Java objects that conform to the JavaBeans API (Applications Programming Interface) and design patterns. Beans can be not only simple components like buttons and sliders but abstract, large and more complex application components as well. A small Bean may consist of a single class; a large Bean may have many.

So, it is obvious that Beans can consist of large and complex applications. Otherwise, these components would not be so popular in current development environments. But the matter in our case, is that the *Visualization Application*, a quite large application, necessitates other tools. These tools are indispensable for the good course of the application. In this way, without the VRML[4] browser and the Web browser, it is impossible to think of working with the application. To a certain extent, these browsers are inextricable from the application but cannot be used within graphical development environments, in order to build complete applications just by connecting prefabricated Java Beans. As a matter of fact, Java Beans are objects intended to be manipulated **visually**, within a graphical application builder.

Of course, we could just have encapsulated the classes of the *Visualization Application* in Java Beans. In this manner, we would be confronted to real Beans, but Beans that cannot be used in graphical development environments. If we build the application in this way, it will not engender a real plus in the **object-oriented**

[3] This is about the application we have developed during our training at the University of Port Elizabeth (UPE, South Africa). Through the thesis, we will refer to this one as the *Visualization Application*. Basically, this application allows the visualization of 3D Charts in a *Virtual Environment*.

[4] VRML is an acronym for Virtual Reality Modelling Language. This language is introduced in the chapter four related to the Implementation of the *Visualization Application*, as well as the notion of browser.

design we have tried to follow. It will just overload the design and the implementation of the application's architecture, by adding new features specific to the Java Beans.

Indeed, it will be required to have a support for **events** allowing Beans to fire events, and informing builder tools about both the events they can fire and the events they can handle. Likewise, it will be needed to have a support for **persistence** allowing Beans that have been customized in an application builder to have their state saved and restored; a support for **introspection** allowing a builder tool to analyse how a Bean works; and a support for **customisation** allowing a user to alter the appearance and behaviour of a Bean.

Using the *BeanBox*[5] application (available at http://java.sun.com/beans/) as the builder environment in which the Beans will be used, we could observe how Beans are more suitable to small independent programs. Therefore, small applications that offer specific and different functionalities and that depend on absolutely nothing, can be easily reused in the BeanBox; effortlessly connected with other Beans; and eventually, all this can form a complete and new application.

Subsequently, to enable the interactivity we will use *Virtual Reality* (VR) technology to allow us to build virtual worlds to visualize information. *Virtual Reality* defines an artificial environment created with computer hardware and software and presented to the user in such a way that it appears and feels like a real environment [Lycos2000].

In reality, as new technologies emerge, different types of computer system become possible, with often more graphical and sophisticated user interfaces. A suite of technologies, such as powerful graphics computers have enabled *Virtual Environment* (VE) interfaces to be realised. However, new technologies must undergo a process of maturity and frequently suffer teething problems. For example, in the early technology-driven phase, a new technology can be difficult to employ and its potential benefits may not be wholly apparent or widely realised [Winograd1995]. Currently, VR appears to be in this phase of maturity.

Loeffler and Anderson [Loeffler&1994] identify *Virtual Environments* as three-dimensional, computer-generated, simulated environments that are rendered in real time according to the behaviour of the user. These environments differ from conventional interface types, bringing new challenges to human-computer interface design.

The ability of VR to simulate three-dimensional objects inside existing or imaginary environments has proven to be useful to education, advertising, business, military, science and entertainment industry. VR facilitates the simulation of realistic representations as well as abstract representations [Darville&2000]. Interactive 3D visualization thus becomes possible by means of panning, zooming, rotating, picking and changing viewpoints. The next figure shows an example of a virtual world: a fish swimming and in front of a skeleton.

---

[5] This application comes with Sun's Bean Development Kit (BDK). But *BeanBox* is by no means a real application builder environment. Its job is to provide a simple reference platform in which we can test our Beans. Quite several examples of Beans are available with this BeanBox.

Figure 2: a Virtual Environment

Of course, this figure gives only a static view of the virtual world. The navigation tool, at the bottom of the figure, allows the user to let swim the fish through the world. We will go back to these technologies later in the thesis.

Temple believes that 3D technology will be used in the future to design multimedia environments [Tebbut1999]. A lot of companies are now working on developing virtual 3D environments, like the CD player of IBM. The IBM Real CD (see Figure 3) is implemented in Java and is available as an application (http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/400).

Figure 3: IBM Real CD player

II. Goal of the thesis

The purpose of the proposed study is validation of Wesson's arguments that Java technology is mature enough to provide the tools for data visualization as defined earlier. The objective will be to show that applications can be rapidly mounted on the World Wide Web that will allow an end user to view the data in a number of different ways, interactively and in real time.

But for this thesis we will focus on the interactive visualization of large data sets and show, by means of a case study, how this can be implemented using chiefly the Java and VRML technologies.

The architecture of the implementation should be object-oriented, in order to facilitate any extension of the *Visualization Application*. We have seen that we will not make use of the Java Beans technology to achieve this aim. Throughout the thesis we will try and show you how we manage to achieve this goal without these Beans, but by other traditional means, like the mechanism of inheritance in Java. In particular, in the chapter three concerning the design of the *Visualization Application*, you will see how we tried to build a generic architecture for the charts' generation. What is more, a lot of attention will be made to making the visualization usable and interactive, especially via the design of the Graphical User Interface.

But, as we will see it later, the interactivity of the visualization could be higher. In fact, the interactivity we manage to propose to the user concerns mainly the Graphical User Interface that precedes and generates the charts' visualization in virtual worlds. The virtual environment proposes also a certain interactivity (via its navigation tool) in order to allow the user to navigate through the world, and to profit from the advantages offered by the VR technology. But an important dimension of this interactivity, concerning the modification of a chart directly in the virtual world, is missing to the *Visualization Application*.

This study is also a continuation of Darville and Van Espen's project (*Study of Virtual Reality Technology Used in the Visualization of Business Information).* They analysed the kinds of graphics (or charts) and their semantics, which can be applied to the visualization of data. We aim to employ their analysis (only a subset of this one) of the semantics of the different charts and also analyse the semantics of the considered data sets.

The following four major steps have been identified at the beginning of our training at UPE in order to achieve our goal, and we will present you, at the conclusion of the thesis, what has been exactly done concerning these steps:

> **(1)** There is a need to identify suitable data sets as an application domain for these ideas. This will also include an analysis of the semantics of these data sets. These data sets should all be characterized by a multi-dimensional nature, which could possibly be graphically represented using VR and 3D technology in a Web environment.

(2) To further investigate Darville and Van Espen's project, we will mount and test their classes and implement several examples. We will also try and apply the Java Beans technology to some components of their architecture.

(3) Additional literature studies (e.g. [Card1999]) will be made on information visualization. We need to categorize the semantic properties of data sets in order to be able to classify those data sets and see which visualization techniques are more appropriate in a certain situation for a certain data set.

(4) An Applet needs to be developed that uses the tools provided. Java will be used to create this applet. As a matter of fact, the use of Java as a development language for the Web is growing rapidly. The user interface is also a key aspect of the proposed interactive application. Current interfaces are still too complex and difficult for the typical user to use effectively [Wessona2000].

In the light of the above, a critical assessment needs to be made of the current (and possibly future) ability of Java to deliver such applications, and of the Virtual Reality to allow such an Interactive Information Visualization. This implies also a utility and usability analysis.

III. Content of the thesis

First, we will present the semantic analysis of the core components of the *Visualization Application*, that is to say the data and the charts. In this chapter, we will introduce the case of UPE to which we were confronted. This case will constitute the base of the application. The categorization of the data set that this university kindly put at our disposal, will also be effectuated.

Then, an assessment of the Interactive Visualization will be examined. The same chapter will also contain a utility and usability presentation and analysis. After that, we will bring you into contact with the concepts of Interactivity and Graphical Visualization. In reference to this, a comparison between two-dimensional and three-dimensional representation will be proposed. The last part will present in detail the *Virtual Environments*.

Followed by, the next chapter will cover the design and the architecture of the *Visualization Application*. The purpose of this chapter is initially to give the specifications and the architecture for the part of this application concerning the generation of different charts, and to propose a prototype for the User Interface.

Of course, as we have built an application, we will present the real implementation of this one. Also, the Implementation tools we have used to achieve this goal will be discussed, notably in a requirement perspective. Examples of the actual implementation will be given as well.

Eventually, a succinct task analysis of the *Visualization Application* and the functional architecture will precede the general conclusion of the thesis.

Chapter One

Semantic Analysis of Data Visualization

I. Introduction

This chapter will present and analyse the semantic properties (restricted to the goals of the *Visualization Application*) of the data and the charts (or graphics). These three-dimensional charts will be visualized within a virtual world, and thus we have also to take this into account.

The case of the *University of Port Elizabeth* (UPE) will also be introduced, and we will explain what we have done in order to prepare the development of the application. So, we will first present the situation in which we found ourselves at UPE, and then the data we have planned to use for the prototype of the *Visualization Application.*

Afterwards, we will analyse the properties of the data with regard to the application's requirements. That means that the analysis will be to a certain extent limitative. In this manner, we will be able to link specified types of data with different dimensions[6].

Next, we are going to take into consideration the particular case of the chosen data set of UPE's students, and will categorize it in relation to the data's semantic analysis.

Finally, we will introduce some different charts, and of course what they allow us to visualize as graphical representations.

II. Presentation of UPE's case

In order to identify suitable data sets for the *Visualization Application*, we were proposed different alternatives from the *Computer Sciences Department* of UPE. Our training took place in this same department, at the *Honours Students Laboratory*. The chosen data set will have to be large enough, and to possess multi-dimensional attributes, as an application area for the project on data visualization.

We investigated three alternative possibilities, and opted for the first one as regards the *Aptitude Test* results. We chose this one because it proved to be the most interesting and the other possibilities were not in such a way suitable for 3D visualization in a *Virtual Environment.*

---

[6] As we will see it later, a dimension is a main characteristic of a chart, especially in a *Virtual Environment*. Each chart can convey a variable number of dimensions, like the three axes (x-, y- and z-axis), the colour, the size and others. The fact to associate a data type with a dimension constitutes the most important part of the task that materializes the *Visualization Application.*

II.1 *Aptitude Test* Results

These test results arise from of the UPE *Placement Team*. Since almost three years, this team of researchers is working on a system, named **Accuplacer**, to establish compulsory entrance examinations. Indeed, till now these examinations do not prevent people who failed for them from entering the university.

At UPE, we interviewed Dr Jean Greyling, responsible of the *Placement Team*. Before this reunion, he had a meeting with the Team to discuss the **Accuplacer** results and their possible representations. So far, they only make use of two-dimensional graphics for their analyses.

The data set contains all personal information about the students, their school results (Mathematics, Science, Biology, and so on) and the *Aptitude Test* results from **Accuplacer Testing System** (Algebra, Arithmetic, English, Sentence, and others).

During the meeting he had with the *Placement Team*, they proposed some possibilities for our project.

First of all, we have the *Successful Profile*. The results of which are represented in 2D-Graphics like the one in figure 4 just below. One curve represents the *Successful Profile* (the blue one), and the other represents the results for a particular student. The letters A, B, C, D represent different tests. And so to decide if this student is able to enter the university with regard to the *Successful Profile*, is not an easy task.

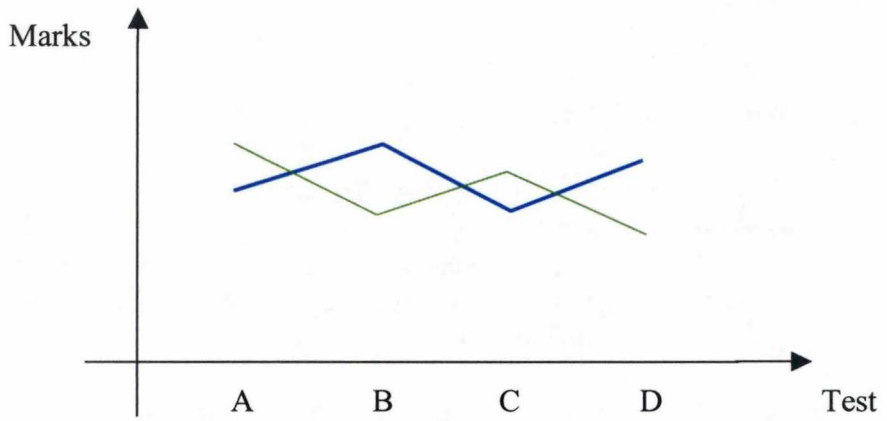

Figure 4: The Successful Profile

A multi-dimensional visualization of those graphics could be very interesting to analyse the results of a student (for example in comparison with the *Successful Profile*) and to classify this one amongst the proficient, the medium proficient or the non-proficient students (classification proposed by the *Placement Team*).

The above graph only considers one student each time. It is also possible to consider all the students at the university, per faculty and per department. Here, we

want to make comparisons, e.g. between January and November results, for a group of students in order to analyse the evolution and the relationships between the results (e.g. compare the percentage of success and failure).

We can also consider all the students of each department, take their *Aptitude Test* results and their university results, and analyse the evolution between different years. In addition, we can take the students of a particular department, and analyse the evolution of their results for the first year, the second year and then the last year.

II.2 Results from Usability Tests

The proposal was to analyse the usability results of these tests performed in the new *Usability Lab* at UPE. The *Usability Lab* was incomplete at the time of this research, so no results were available for visualization.

II.3 University of Port Elizabeth Student Profile

We interviewed Morne Streicher, junior lecturer at UPE, who is researching a *User Model* for students at UPE. He has a lot of interim data currently available but this is very similar to the *Aptitude Test* results.

We therefore decided to choose the *Aptitude Test* results from the **Accuplacer** system as our application domain for the prototype we are going to create (i.e. the *Visualization Application*). That means we will make use of the database containing the *Aptitude Test* results, other school results and personal characteristics (like the gender, the race, the faculty, the age and so on) to build the charts that will be visualized within virtual worlds.

III. Data Categorization for the Visualization Application

The data categorization implies other semantic aspects than the ones we will develop under here. So, Darville C. and Van Espen S. (in [Darville&2000]) in their *Introduction to Semantic Properties of Business Data* have analysed the following properties of the data in the perspective of their visualization objectives : the *data organization*, the *data type*, the *temporal dimension of data* and the *data set size*.

We totally agree with their analysis, but in the *Visualization Application*'s point of view, we just need to deepen the semantic property with reference to the *data type*. Only this one is necessary to build dynamically a chart, and besides the *data organization* (geometric and non-geometric) is implicitly included into the type of the visualized chart.

III.1 The different dimensions and the data types

In order to categorize the data set, we will use the terminology that [Card2000] employs. In this book, they talk about *"three basic types"* for data : **nominal**, **ordinal** and **quantitative**. The two first are also often referred to as **qualitative** data.

- A **nominal** (N) variable is an unordered set, such as film titles {Goldfinger, Ben Hur, Star Wars}. The nominal data types are so unordered collections of symbolic names without units.

- An **ordinal** (O) variable is a tuple (ordered set), such as film ratings <G, PG, PG-13, R>, or the sequence of calendar months' names (i.e. from January to December). The ordering does not reflect the magnitudes of the differences.

- A **quantitative** (Q) variable is a numeric range, such as film length [0, 360]. The quantitative data types are so usually expressed as *real* values in the data set. The precise numerical value has a certain importance in the semantics of the data. A typically quantitative data set is the length of objects.

The nominal variables are only equal to or different from other values. The ordinal ones obey a '<' relation. And we can do arithmetic on the quantitative ones.

Let's look at the different ways to visualize data we have chosen (all this will be recapitulated in Table 1). We have the different axes of a chart. In most cases, these consist of an x-axis, a y-axis and a z-axis. These axes can always be used to represent nominal, ordinal, or quantitative data. But they are more suitable to convey quantitative information in comparison with the other dimensions we will discuss below.

Besides, we can employ colour, shape, size of the shapes, texture and transparency as dimensions for the different charts. The colour dimension has to be used with nominal data and cannot be used with ordinal, because this dimension represents an unordered set of values. The same case applies to the shape and the texture dimension.

It is more difficult to classify the transparency dimension. Normally, it would have to be used with ordinal or quantitative data. But if it was used with quantitative data, it would mean that we would be able to determine, for example, if a data is two times more transparent than another one. Since that would prove to be difficult, we will not use transparency with quantitative data. Transparency is likely to be more effective with really small sets. Although it is a dimension for ordinal data, we may use it too with nominal data that have a very small set if the other *"nominal dimensions*[7]*"* are no more available (e.g. binary data like the gender variable).

In [Card2000] again, they have divided colour into its three components:

- **hue** ("colours" of rainbows, or in other words, the spectrum of colours),
- **value** (brightness/lightness), and
- **saturation** ("paleness" of colour is lack of saturation; it indicates how much gray has been added to the pure colour).

Hue refers to the specific individual pure colour on the nuance wheel. The **value** of a colour refers to how light or dark it is and is directly related to the gray scale. This value change is created by adding either white or black to the original colour. And the amount of pure hue in the colour refers to its **saturation**. The more the pure hue, the higher the saturation and vice versa.

Hue, saturation, value may be varied independently or in connection to each other. We can so use the value field as a dimension, and in this case, it can be used for ordinal data (e.g. from the brightest to the darkest values). The same applies with saturation. We will use value and saturation in a similar way we use transparency.

For the hue, we may only connect it to a nominal data. Indeed, this term just refers to a pure colour, so neither ordering nor comparison is possible between different values. For instance, it will be very appropriate to associate a specific colour (the hue component) with each faculty, or with each gender (male and female).

To have a better idea of the Hue-Saturation-Value (HSV) subdivision, let's take a look at the next figure.

[7] We mean by *nominal dimension*, a dimension that can be associated with a nominal data.

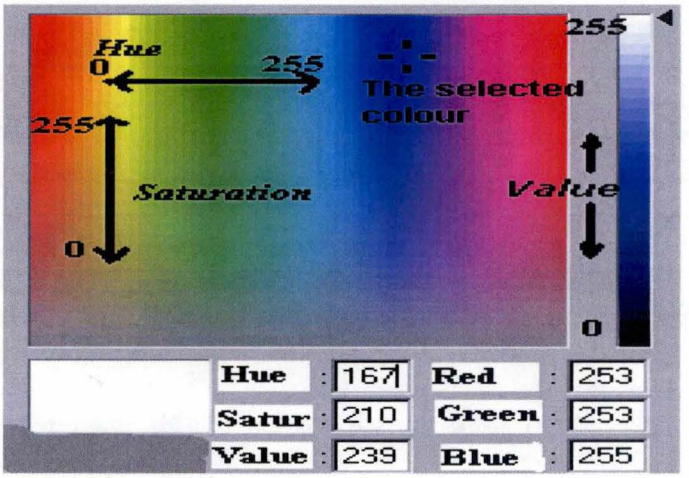

Figure 5: HSV Definitions

We can distinguish on this figure the selected colour. Its hue value is 167 (**H=167**), its saturation value is 210 (**S=210**) and the last one is 239 (**V=239**). The minimum value for these fields is 0, and the maximum is 255. The value of the colour is here 239 and, as we can observe it at the right of Figure 5, it almost corresponds to the lightest value on the scale. The variation of the hue corresponds to the variation between individual pure colours. And the variation of saturation corresponds on the figure to an up and down variation.

In Table 1, we analyse the link between the different dimensions and the data types. For example if we consider the first row, we can say that the colour (more exactly its hue component) dimension will be only used to convey nominal information.

		Data Types		
Dimension		**Nominal**	**Ordinal**	**Quantitative**
Colour	Hue	✓✓		
	Saturation	✓(very small set)	✓✓ (very small set)	
	Value	✓(very small set)	✓✓ (very small set)	
Shape		✓✓ (small set)		
Size				✓✓
Texture		✓✓		
Transparency		✓ (very small set)	✓✓ (very small set)	
Geometrical Position (axes)		✓	✓	✓✓
Orientation				✓
Time				✓

✓ ✓ Suitable
✓ Possible

Table 1: Data type/Dimension Association

Now, we will bring to an end the explaining of this table's content. First, we have associated nominal and ordinal data with the geometric position dimension, because in some cases, we do not have other possibilities for the representation. It will be so for example for the *3D BarChart* (cf. point IV.1).

Then, we can consider the different components of the colour dimension:

- Hue is of effective use for nominal data types. We can for instance associate colours like blue with ordinary values, and red for dangerous values. Associate the pink colour with female persons, and blue with male could also be very appropriate and evocative. The two next figures (Figure 6 and 7) illustrate this point. Figure 6 gives the legend of the chart displayed on Figure 7. We have associated the pink colour with female students, and blue with male students. The average of a student is represented by the size of the sphere associated to this student.

- Saturation is more suitable for ordinal data types, and especially small data sets. But we should remain careful when interpreting saturation and value (or brightness) independently.

- The brightness is not used for quantitative data types. Indeed, the absolute brightness and the change of brightness cannot be perceived linearly.

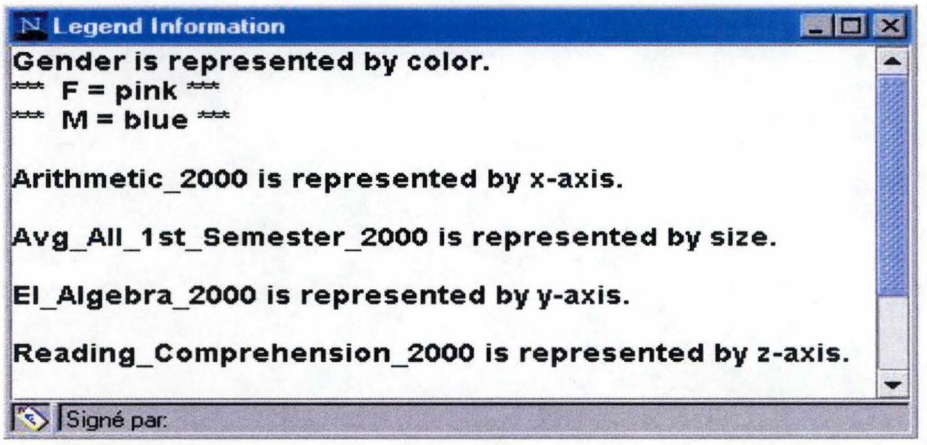

Figure 6: Legend of next figure's ScatterChart[8]

The figure 8 gives a front view (via the navigation tool provided by the VRML browser[9]) of the same chart, as if it was a two-dimensional (x- and y-axis) graphic. The distinction between male and female students cannot be more straightforward.

---

[8] The notions of *ScatterChart* and *BarChart* are introduced later in this chapter (section IV).

[9] The navigation tool and the VRML browser will be defined and explained later in the thesis. But in a few words, the navigation tool allows the user to navigate within the virtual world, and the VRML browser allows the visualization of such worlds in a two-dimensional Web browser (like Netscape Communicator).

For example, one can easily remark that in average, the male students struggle through better in Algebra and in Arithmetic than the female.

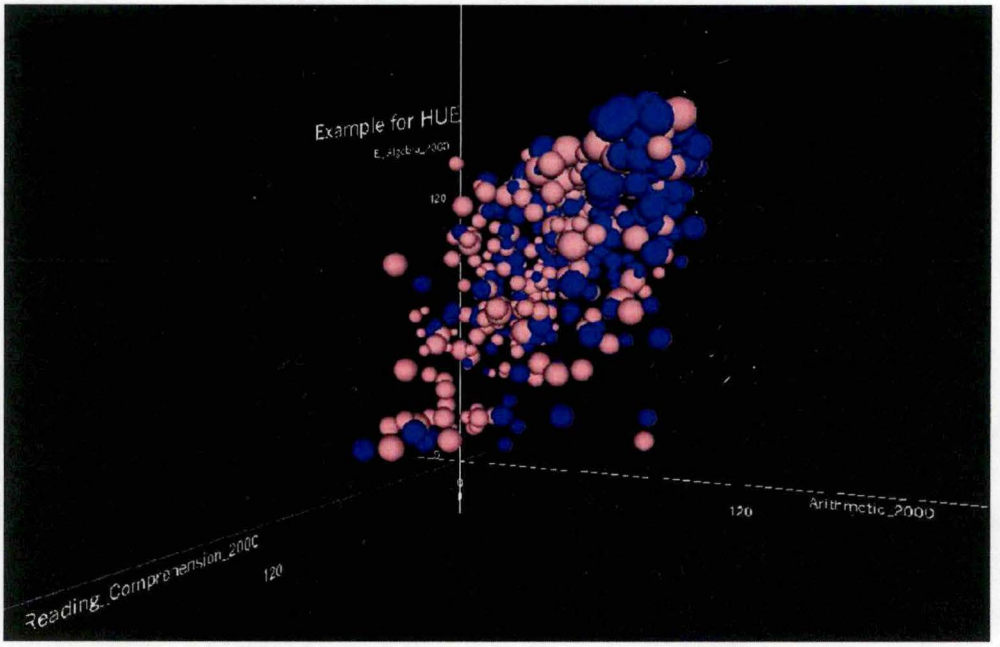

Figure 7: Illustration of the Hue dimension

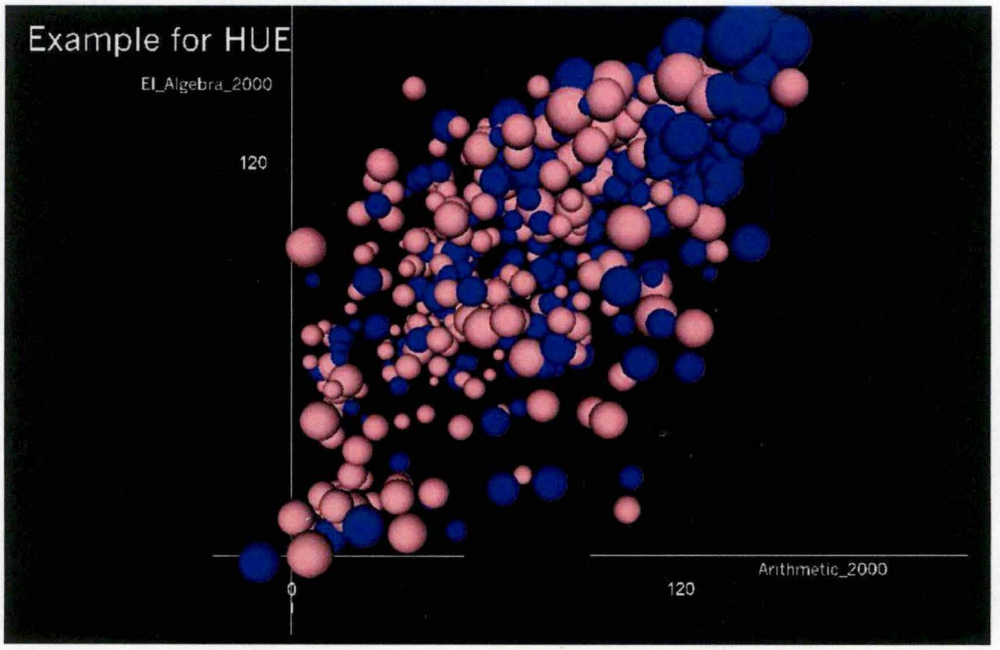

Figure 8: Illustration of the Hue dimension (front view)

The size of the shape(s) as expected will convey quantitative data. We must keep in mind that the size and the shape dimensions are in conflict. If we have more than one different shape in a chart and we want to use the size as a whole dimension for the shapes, it will be difficult to perceive the different sizes of the different figures.

The following example will exemplify this point. Once again, we have chosen to visualize a ScatterChart. The marks in Arithmetic, Algebra and Reading Comprehension are respectively conveyed by the x-axis, the y-axis and the z-axis. The blue shapes represent male students, and the pink ones represent female students. The Average for the first semester 2000 is represented via the size of the shapes, and we have associated the possible shapes with the existent faculties (a box for the students from the faculty of Arts and of Education; a cone for the Economic Sciences and Law; a cylinder for the Health Sciences; and a sphere for the faculty of Science).

The figure 9 gives a front view of the ScatterChart displayed in the virtual world. We have used the same subset of students as in the previous example. But now we also represent on the graphic the faculty of each student (via the shape dimension). The figure 10 gives a more detailed view of the same chart: we have zoomed in to the centre of the chart in order to better distinguish the size of the different shapes.

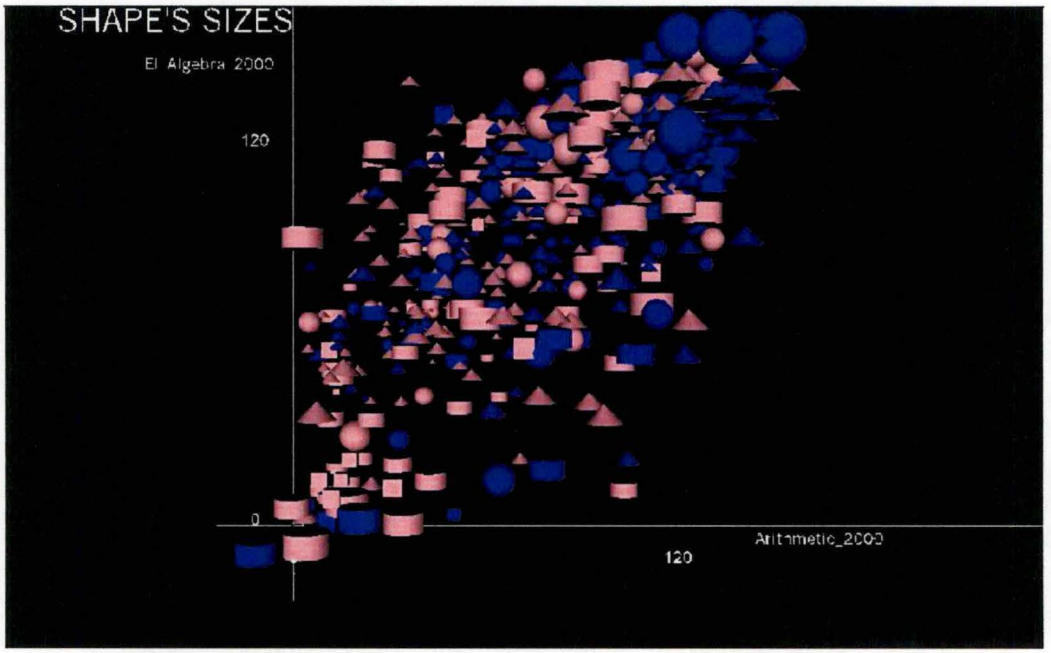

Figure 9: Illustration of the Size/Shape conflict

If we take a look at the next figure we can clearly distinguish the four different shapes (sphere, cone, cylinder and box). But if the user wants to perceive visually the size dissimilarities between different shapes, it will be almost impossible (except when the dissimilarity is high enough). Fortunately, we overcome this problem by allowing the user to obtain the exact values for each object : he just needs to drag the mouse on the looked-for object. So, he can have at his disposal the exact sizes of each shape.

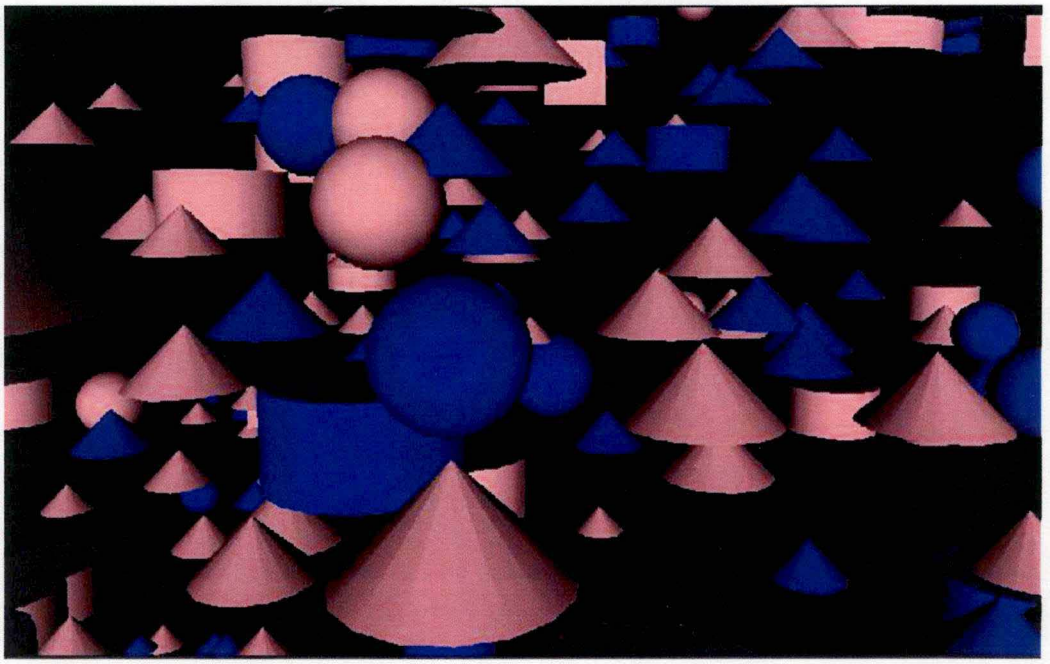

Figure 10: Illustration of the Size/Shape conflict (zoom-in)

Next, the texture dimension presents the same basic characteristics as the colour (hue). So its use will also be exclusive to nominal data types. Here also there is a possible conflict between colour and texture. In fact, most textures do not allow the user to perceive the colour of an object. That's why in the *Visualization Application* the user will have to choose between the texture and the colour dimension for a possible association.

Eventually, even if each variable belongs intrinsically to a specific type, this membership is not rigid. In many cases, the type of a variable can be modified. In fact, its type can always be downgraded and in some cases, it can also be upgraded from the ordinal one to the quantitative one. We may use this transformation in UPE's data set, for instance if we transform the *date of birth* variable to a new *age* variable. The quantitative type of *date of birth* variable becomes ordinal for *age*. The following diagram, Figure 11, illustrates this.

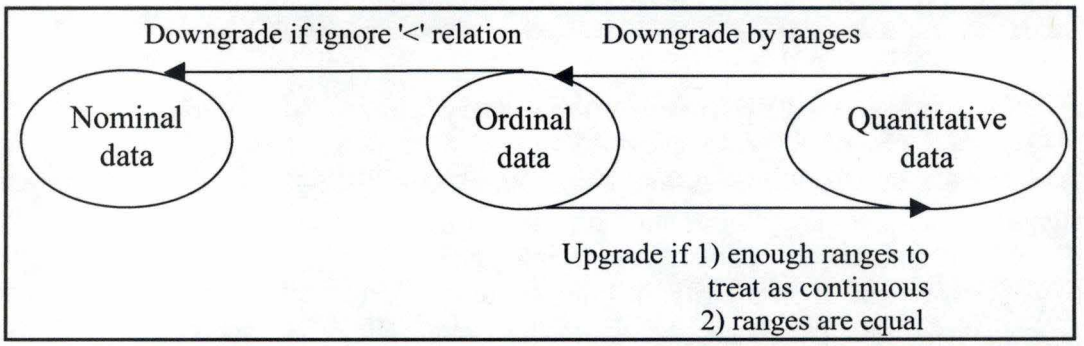

Figure 11: Type Modification Diagram

III.2 Categorization of the *Aptitude Test* Results' data set

First of all, we have the *student number* variable. We could believe that it is a quantitative variable, but this is not the case. It is only a nominal variable. This variable was simply created to give all the students a specific value and different from the others. Indeed there is no "<" relation between the students, compared with their *student number*.

The *surname* and the *name* variables are also nominal. But we don't think they will be used in the charts (or just for user's information and not as dimensions).

The *gender*, the *home language*, the *ethnic group*, the *race*, the *degree* and the *faculty* all are nominal variables and cannot be transformed into ordinal ones.

We have also the *date of birth* variable. Instead of *date of birth* we can use an *age* variable (cf. Figure 11). The *age* variable is a quantitative one, but in most cases we will transform it into an ordinal one. So we will have different categories of *age*.

Next, we have figures that represent all the **Accuplacer** results for the academic years 1999 and 2000, and the average of the first semester 2000. Those are quantitative variables. We could also categorize one of these data to obtain an ordinal variable (e.g. if we want to form two categories: less than 50 and more than 50).

Finally we have the number of courses a student has registered for and the number of courses in which he has passed the examinations. Those are also quantitative variables.

A summary of this is given in Table 2. And for the last column, please have also a look at Table 1.

Variables	***Variable Types***	***Dimensions to visualize the variable***
Student Number	N	Unlikely to be visualized (except if used in pop-up labels[10])
Name and Surname	N	
Degree	N	
Gender	N	Shape, Texture, Colour (Hue), and Transparency
Matric Year	O	Transparency, Value, and Saturation (or Shape, Texture and colour because this variable represents a small set)
DOB -> Age	Q -> O	Value, Saturation, and Transparency
Home Language	N	Colour, Texture, Shape (not enough different shapes to represent differently all the languages[11])
Race / Ethnic Group	N	Colour, Texture, Shape (not enough different shapes to represent differently all the races or ethnic groups)
Faculty	N	Colour, Texture, Shape (not enough different shapes to represent differently all the faculties)
Accuplacer Results and average	Q	Geometric position (x-, y- and z-axis), and Size
Number of courses registered for and passed	Q	Geometric position (x-, y- and z-axis), and Size

Table 2: The data set categorization

[10] A pop-up label explains to the user what the object, on which the mouse cursor is located, represents. In the *Visualization Application* we only use the status bar (at the bottom of the Netscape Communicator window) to convey this information because such labels are not available in the virtual world.

[11] In the *Visualization Application* we only use four different shapes. As there are more than four different languages, if the user wants nevertheless to associate the shape dimension with this variable, he has to link different languages to the same type of shape. The same remark applies to the race, the ethnic group, and the faculty variables.

IV. Some charts and what they allow us to visualize

IV.1 The 3D BarChart (or Column Chart)

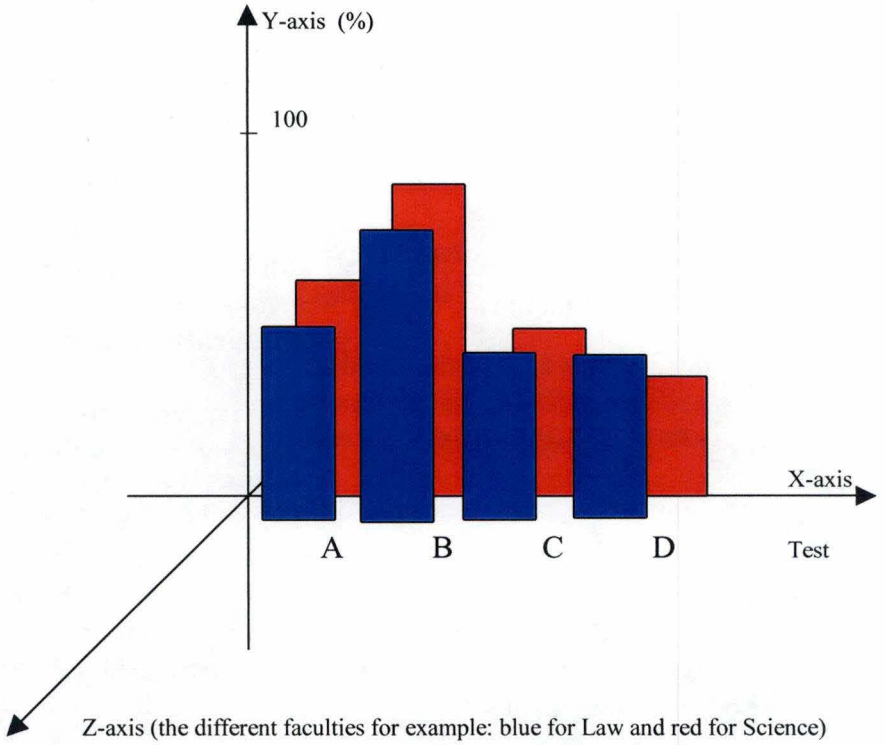

Figure 12: A front to back BarChart

Visual description :

A Column Chart displays nominal (and ordinal) and quantitative information by means of a series of geometric shapes such as rectangles, cylinders, or others. Each column represents a data element, and a complete set of columns represents a data series. Columns may be lined up side to side as well as front to back. It shows values of a single quantitative variable for multiple separate entities. Spacing between adjacent bars should typically be less than the bar width to facilitate comparisons between bars [Preece1994].

Function :

Column Chart is particularly efficient in the following roles :

- Comparing multiple items at various points distributed along a time period,
- showing how relationships between multiple items change with time,
- looking for relationships between multiple data series, and

- condensing onto one graph either several single series already displayed in three-dimensional charts or data series of which visualization could require several two-dimensional charts.

Limitations :

Even if there is no technical limit concerning the number of data series or the series length displayed by the graph, one can observe practical limits above which the graph becomes confusing.

Considering the number of data series generally distributed on z-axis, one can remark in most cases that the magical number [Miller1956] (= **7 ± 2**) is an appropriate limit. The magical number expresses the cognitive limit of the human mind. It means that more than nine data series will worsen charts readability.

The length of the data series depends on the number of data series displayed on the chart and is limited by the necessity to keep a goal clear and not confused view of the chart. This way, the less data series are displayed, the longer these series may be, and vice versa.

Since the BarChart is a VR chart, this limitation is pushed back thanks to the possibilities of interactivity (e.g. drilldown, filtering, sorting, zooming, rotating and so on). Interaction may allow to generate clear and not confused subsets of the graph which focus on particular details.

Application :

So the x and z-axis can be used to represent ordinal or nominal information, and the y-axis should be a quantitative axis.

For example, we can plot on the x-axis the race variable, on the y-axis the average for Arithmetic and on the z-axis the different categories of students according to their faculty, degree, language, age (this quantitative variable has to be transformed to an ordinal one before being used here), race or ethnic group. This example, with the faculty variable plotted on the z-axis, has been applied to the *Visualization Application*, and the result is a 3D BarChart built within a virtual world. The figure 13 gives a general viewpoint of the generated virtual chart.

Eventually, if the cardinality of the different categories is less than three (e.g. the gender variable), then it is preferable to use a 2D BarChart where two columns are located beside each other.

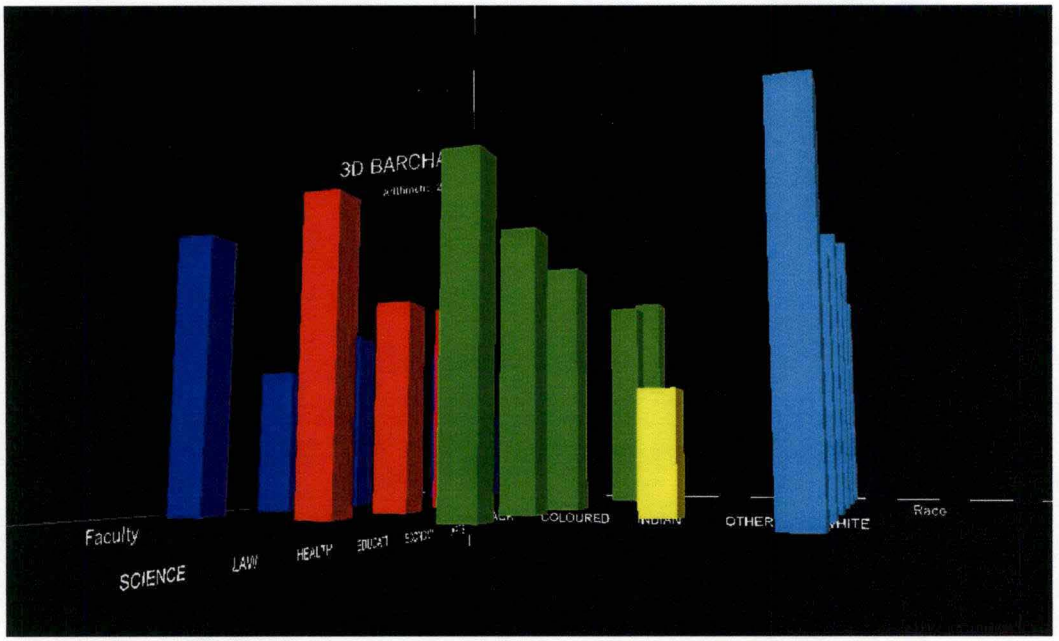

Figure 13: A Virtual 3D BarChart

IV.2 The ScatterChart

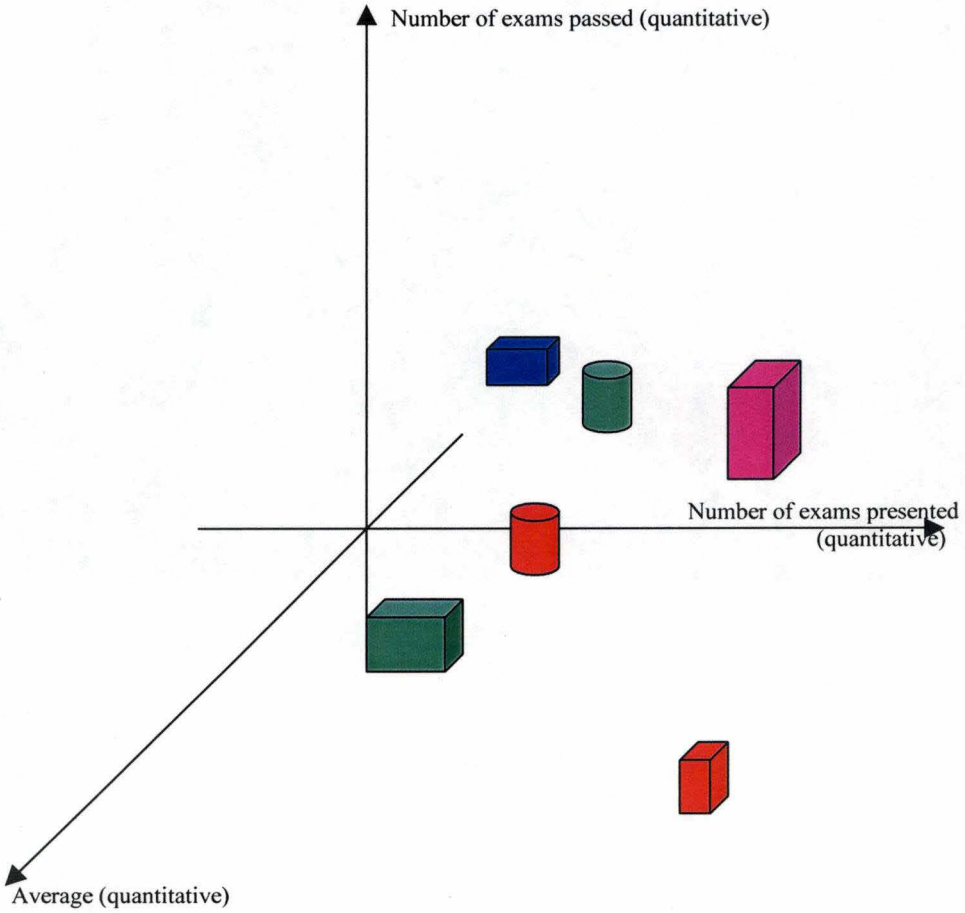

Figure 14: A ScatterChart

Visual description :

> "Also called *Point Chart*, the ScatterChart displays discrete quantitative information by means of points (represented by spheres, cubes or different other shapes placed in the chart). A three-dimensional ScatterChart generally has quantitative scales on all three axes". [Preece1994]

The ScatterChart possesses seven dimension to display information. These dimensions are :

- the three coordinates (x, y and z),
- the colour or the texture
- the transparency
- the shape and
- the size of the shape.

Function :

A ScatterChart is used to correlate multiple data in order to identify trends, make comparisons and find connections hidden within data.

Limitations :

As its function is only to spot the presence of outlying data within a set, it seems that this charts offers a more flexible limit on the number of data series or set on the graph than the other charts.

Application :

Besides the three quantitative data represented by the axes, four others seem relevant. These are the gender (nominal data), the home language (nominal data), the ethnic group (or the race : both are nominal data) and the faculty (nominal data).

We must represent these data by four new dimensions (if we want to represent all of them). The choice could be the following:

- Gender can be represented by transparency or shape;
- Home language by colour or texture;
- Ethnic group (or race) by colour or texture; and
- Faculty by shape, colour or texture.

This graph is useful to provide an overview. It enables us to have an overall picture of all the students and makes it possible to detect possible correlations between these various dimensions.

It is possible to restrict the number of data to display in order not to overload the graph or to make it possible to concentrate on some particular data. For example, we can display only the results of students belonging to a same faculty.

Some patterns can also be expressed more efficiently by plotting one of the five dimensions on an axis with the costs of one of the three currently represented. That could make it possible to better highlight some correlations.

Another Chart could be interesting. It is a variant of the *Ribbon Chart*[12]. Ribbons may be used to join some points of the ScatterChart in order to highlight the relationship between them. This new chart is called *Scatter Ribbon Chart*. It benefits from the advantages of the *Ribbon Chart* as well as the ScatterChart: it shows a global evolution and it draws user attention to some more important data [Darville&2000]. The x- and z-axis can be used to represent ordinal or nominal information. And the y-axis should be a quantitative axis.

---

[12] The *Ribbon Chart* displays continuous quantitative information by means of thick lines, visually similar to ribbons. Typically, it has a quantitative scale attached to the vertical axis and a qualitative or quantitative scale on the other two axes depending on the data nature. This chart is presented more accurately in the chapter three *Design of the Visualization Application* (a graphical representation of the chart is also available).

For example, we may put on the x-axis the names of the Accuplacer tests, on the y-axis either the results obtained by a student or the percentage obtained by a group of students for those tests and on the z-axis the different categories of students according to their faculty, language, age (this quantitative variable has to be transformed to an ordinal one before being used here), race or ethnic group. The points obtained by the same student or the same group would be joined. We could also compare the results obtained by a student with the *Successful Profile* (cf. section II.1).

Now we are going to apply a scenario to the *Visualization Application* in order to create a virtual ScatterChart. Arithmetic, Algebra, Sentence Skills and the Average for the first semester 2000 (four quantitative variables), are respectively conveyed by the x-, the y-, the z-axis, and the size dimension. Race is represented by colour: gray for Black people, orange for the Coloured, cyan for the Indian, and white for the White. Gender is represented by transparency: opaque objects are female students, and semi-transparent objects are male students. At last, the shape dimension is used to convey the different faculties: a box for the faculties of Arts, Education and Law; and a sphere for the faculties of Health Sciences, Economic Sciences and Science. In this example, we have also decided to restrict the number of students for the graphic. Indeed, only 227 students are projected in the chart: the restriction is that each student has more than 50 for the three tests and also for the average.

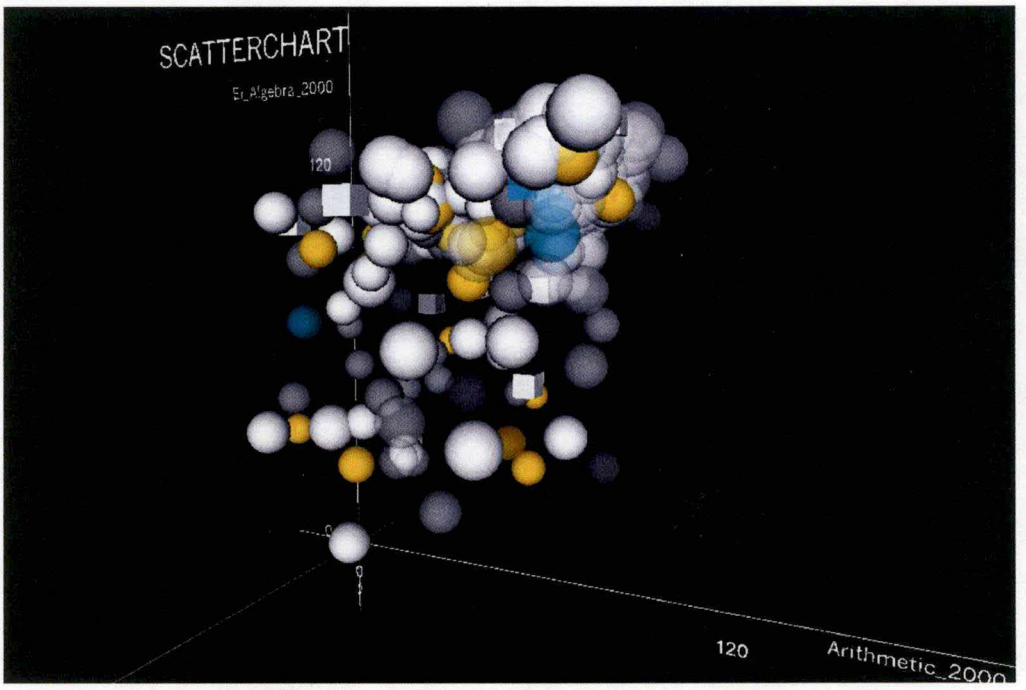

Figure 15: A Virtual 3D ScatterChart

IV.3 The Temporal Star

A *Temporal Star* is a set of *Simple Stars* linked by a central vertical axis representing time [Noirhomme00].

A *Simple Star* is a circular graph with three or more radial axes distributed equally around the 360 degrees of the circle and representing statistical variables. All axes have either the same scale or a specific one. Data elements are plotted on the axes. A quantitative variable (average or interval) is represented by a graduated axis and a qualitative one is represented by dots equally distributed on an axis [Darville&2000].

Here the vertical axis represents the time. But we cannot represent it as a quantitative variable. It has to be an ordinal one, for example, the different years since 1996. But in the database as temporal values we only have the tests results of 1999 and 2000.

We can also use this axis to convey other ordinal information (e.g. the age variable transformed to an ordinal variable). And if we want to compare some simple stars between different categories (e.g. between degrees), the vertical axis can be used as a nominal one, even if initially this axis represents a progression.

A *Simple Star* allows about 16 variables (i.e. 16 axes) to be displayed. And even if this number can be increased, progressively, it will lead to an overload of the graphic [Darville&2000].

The axes of a *Simple Star* will generally be used for quantitative variables (e.g. the *Aptitude Test* results), but they can also be employed for ordinal and nominal ones.

The next figure shows an example of *Temporal Star*, just to give you an idea about how such a chart looks like. We have obtained this graphic via the ISO-3D tool. ISO-3D is a acronym for Interpretation of Symbolic Objects in 3D representation, and it has been developed under the direction of Professor Monique Noirhomme[13]. The vertical gray axis in the middle of the image may represent the time evolution, and there are three *Simple Stars*, with nine radial axes.

Through lack of time, we were not able to extend the *Visualization Application*, in order to include the *Temporal Star* chart, as well as the *3D Tree* (cf. next paragraph). But we do really believe in the potential of such graphics in a *Virtual Environment* for a data analyst. Thus, a future research in this field could attempt to treat the dynamic generation and virtual visualization of such graphics.

---

[13] Prof. M. Noirhomme is a lecturer at the University of Namur (*Facultés Universitaires Notre-Dame de la Paix).*

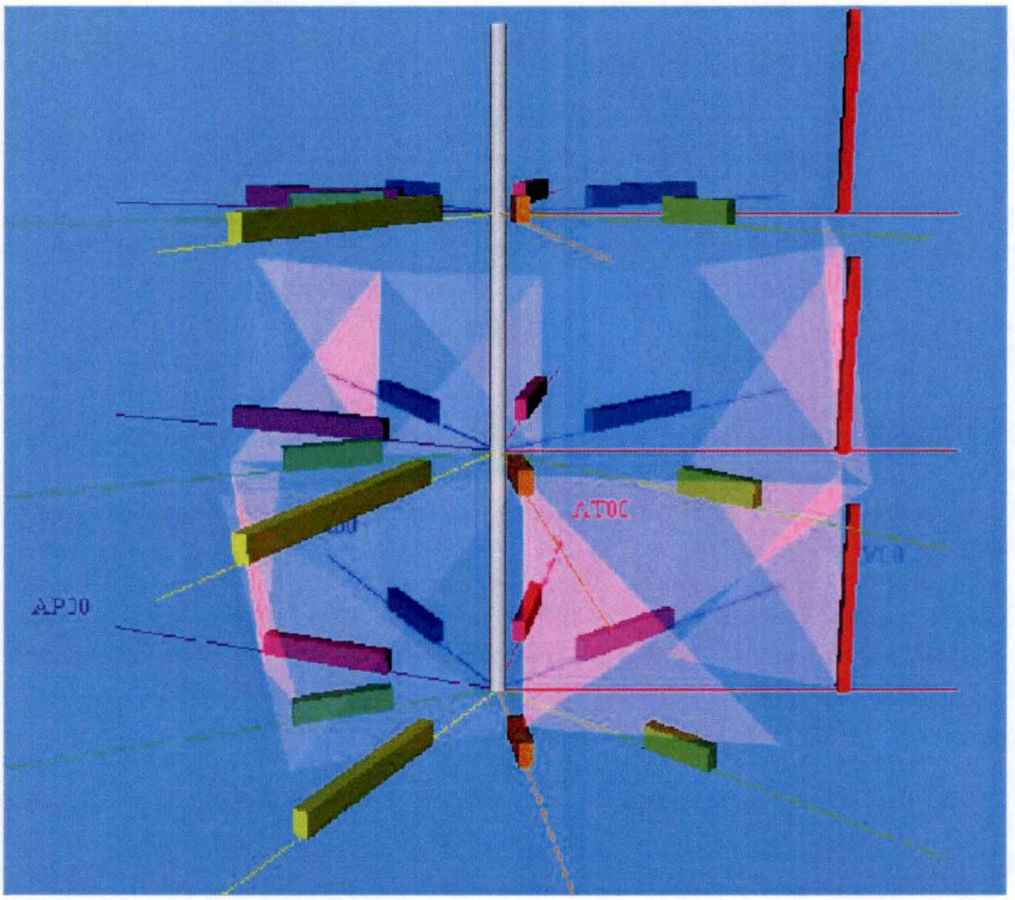

Figure 16: A Temporal Star

IV.4 The 3D Tree

This chart represents hierarchical data. There is a specific relation between the nodes. The figure 17 gives an outline of this specific chart.

We can use it if we want to have an overview of all the students. For example, the root will represent all the first year students at University of Port Elizabeth. And then we can use the different categories (degree, faculty, language, age group, race and ethnic group) to create new children nodes, until there are no more categories left.

Each node will be a predefined shape. And we can use the size of those shapes to represent, for example, the proportion of success between brother nodes. Each one of these nodes could also contain a specific chart to visualize the information embedded in it (cf. Figure 18).

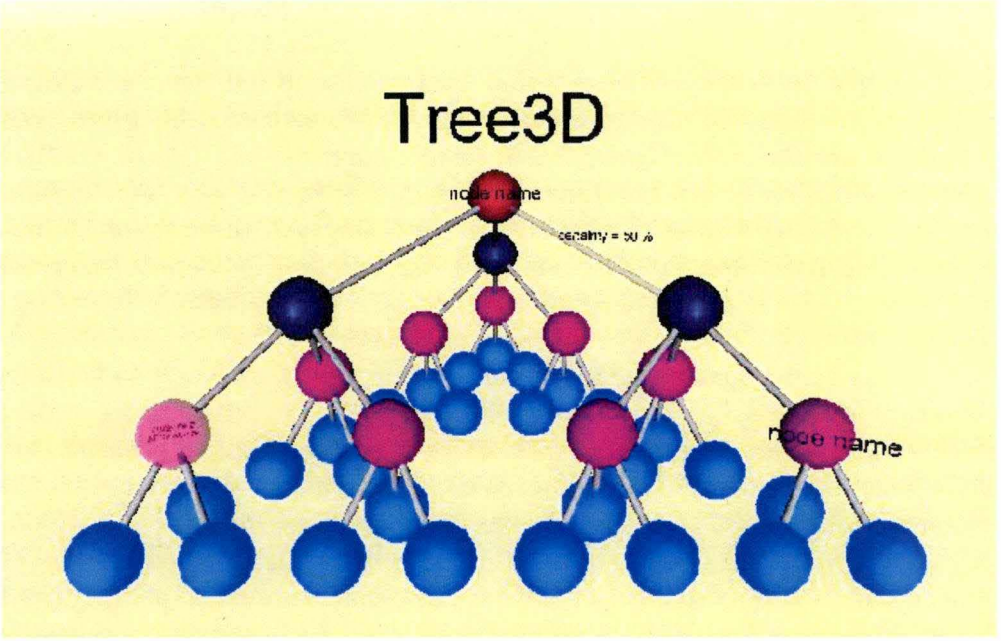

Figure 17: A 3D Tree graph

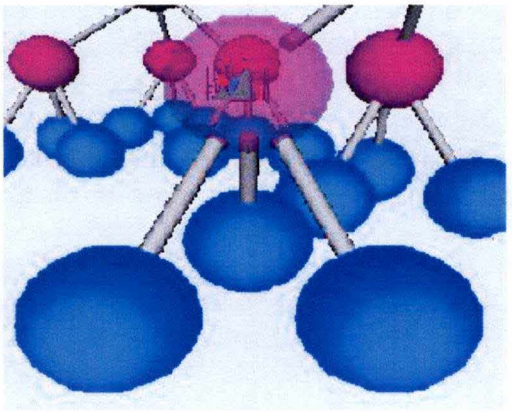

Figure 18: An embedded chart in a node

IV.5 Conclusion

Our attention had to focus only on some charts. Because there are a lot of possibilities other than the four presented here. For instance, the other most common charts are the *Area Chart*, the *Surface Chart*, the *Pie Chart*, the *Map Chart*, the *Combination Chart*, the *Animated Chart*, the *World Chart*, the *InfoBall*, the *Giotto 3D*, the *Information Cube*, and the *DataCube*. All these graphics have been presented in [Darville&2000] (section 3.4: *Presentation of the most common charts*).

The first choice was the *ScatterChart*. In fact, this chart allows the user to visualize all the students, if necessary. Besides each object on the chart represents a student. As we aim to provide the user with an *Interactive Visualization tool of* ***Large*** *Data sets*, this choice is especially appropriate. We can also advance the fact that this chart tolerates easily the use of six dimensions and more if possible.

To offer a certain flexibility to the user, we have also concentrated on another chart, the 3D BarChart. This one is more general, and gives the user an overview. In addition, this chart is extremely well-known and also its properties. But we can use on this one only three dimensions. In our case study, we have added one dimension (the colour) but its objective is only to give the user a better perception of the chart, and to allow him to distinguish the different columns.

A future work could certainly extend the application in order to include more charts, and mainly the Temporal Star and the 3D Tree, because their semantics is quite different from the two previous ones. Thanks to the Temporal Star, the user can make comparisons between the same data along the time. And the 3D Tree should show all the detailed information for a particular node if the user selects this node, and from detailed information be able to show results for groups (zoom in/out facility).

Chapter Three

Assessment of Interactive Visualization

I. Introduction

In the thesis, we treat large data sets. Therefore, a user request can provide hundreds of potentially useful items. Our hypothesis is that interactive 3D techniques can be useful to help the user understand, analyse and filter such results. Of course we need to provide usable and useful tools to the users or else the interactive 3D techniques we will propose would be ineffective.

In this chapter, we are going to introduce the concepts of *utility* and *usability*; with regard to the *Visualization Application* we want to create. Then we will see how interactivity and graphical visualization can assist us in reaching one of our goals, namely the *Interactive Visualization* of large data sets. Next, we will justify the choice of three-dimensional representation, by comparison with two-dimensional representation. Eventually, the fifth section of this chapter will introduce and define the concept of *Virtual Environment* (VE). In this same section, we will compare VE interfaces with Direct Manipulation interfaces. It will also be a matter of presenting the benefits of the visualization in such environments, and the common problems encountered in this field.

II. Utility and Usability

Utility is a measure of how well the system helps the user fulfil one or more real world tasks, and asks a simple question: will the system do what is needed functionally? *Usability* is a measure of how well the user can interact with the system and meet their goal. It means the capability to be used by humans easily and effectively. Both concepts depend upon the design of the tool in relation to the users, the tasks and the environments, and upon the success of the user support provided (training, manuals, and other job aids such as off-line 'help' facilities). It is important in the overall success of a system in that it affects how well the user can carry out their task or meet goals when using the system. For voluntary-use systems, such as the *Visualization Application*, *usability* is also important in affecting the user's motivation to use the system.

This meaning of *usability* complies with the definition proposed by Brian Shackel and Simon Richardson [Shackel91]:

> "the usability of a system or equipment is the capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfil the specified range of tasks, within the specified range of environmental scenarios".

Utility and *usability* are also often referred by the concept of *usefulness*, as introduced by [Nielsen98a]. *Usefulness* is the issue of whether the system can be used to achieve some desired goal, and this can be divided into *utility* and *usability*.

> "The **usefulness** of a system is determined by two components:
>
> **Utility**: Does the system do anything that people care about? If the system does something irrelevant or if it doesn't solve the main problem, then it does not matter whether it is easy to use: it will be a poor system in any case.
>
> **Usability**: Can the user use the system and can he or she do so effectively? Even if the system does exactly the right thing in theory, it will still be a poor system if the user cannot figure out how to get it to work."

In reference to the *Visualization Application* we have created, the range of users, the possible training and the range of tasks will be discussed in the chapter related to the Task Analysis[14]. In this task analysis we will study the users behaviours to create in order to produce a useful and usable system.

To achieve the goal of usefulness, we also need to focus on the human-computer interface (HCI). For many ordinary users and even for some computer professionals the HCI is still more of a space frontier and a time barrier than an open door to communication. By focusing on the HCI, we focus on the most important component in a human-machine system, that is to say the human factor. The other components are the task, the tool and the environment. So we must see the user as the centre of the computer system instead of a mere peripheral. Otherwise, as stated by Shneiderman in [Shneiderman92], users may have to cope with frustration, fear and failure as a result of being faced with excessive complexity, incomprehensible terminology and chaotic layouts in a system.

According to Ping Zhang [Zhang99] :

> "Any decision-making support system or problem-solving support systems should be developed from a human-centred perspective. A human-centred visualization system should therefore help the user to achieve cognitive effectiveness and efficiency by shortening cognitive distance from visual representations and removing mediation for thinking. It's a system designed from the outside (i.e., the user) and not from the inside (i.e. the hardware) ; a design philosophy that emphasizes the needs and abilities of the user, which can be opposed to a philosophy that concentrates on efficient computation."

To allow the evaluation of the *utility* and *usability* of an HCI, [Nielsen98a] has proposed five different criteria:

1. **Ease of learning**: How fast can a user who has never seen the user interface before learn it sufficiently well to accomplish basic tasks?

---

[14] This chapter gives a succinct presentation of the task analysis concerning the *Visualization Application*, as proposed by Professor F. Bodart in [Bodartb1998].

2. **Efficiency of use**: Once an experienced user has learned to use the system, how fast can he or she accomplish tasks?

3. **Memorability**: If a user has used the system at some earlier date, can he or she remember enough to use it more effectively next time (or does the user have to start over again learning everything every time)?

4. **Error frequency and severity**: How often do users make errors while using the system, how serious are these errors (burning down a cement plant is worse than getting the wrong player's score on a golf site), and how easy is it to recover from a user error?

5. **Subjective satisfaction**: How much does the user *like* using the system?

These criteria will be used to evaluate the *utility* and *usability* of our interface, in the task analysis. The next figure (Figure 19) shows the place of *utility* and *usability* in the overall acceptability of a system. It summarizes the different notions presented in this section. The components of the system acceptability we have to focus on are those related to the *usefulness*. In fact, the other factors are less important or inappropriate with regard to the goals and content of the *Visualization Application*: the social acceptability of the application is beyond all doubt, and the components of the practical acceptability (except the *usefulness*) do not seem pertinent enough to be studied in this thesis.

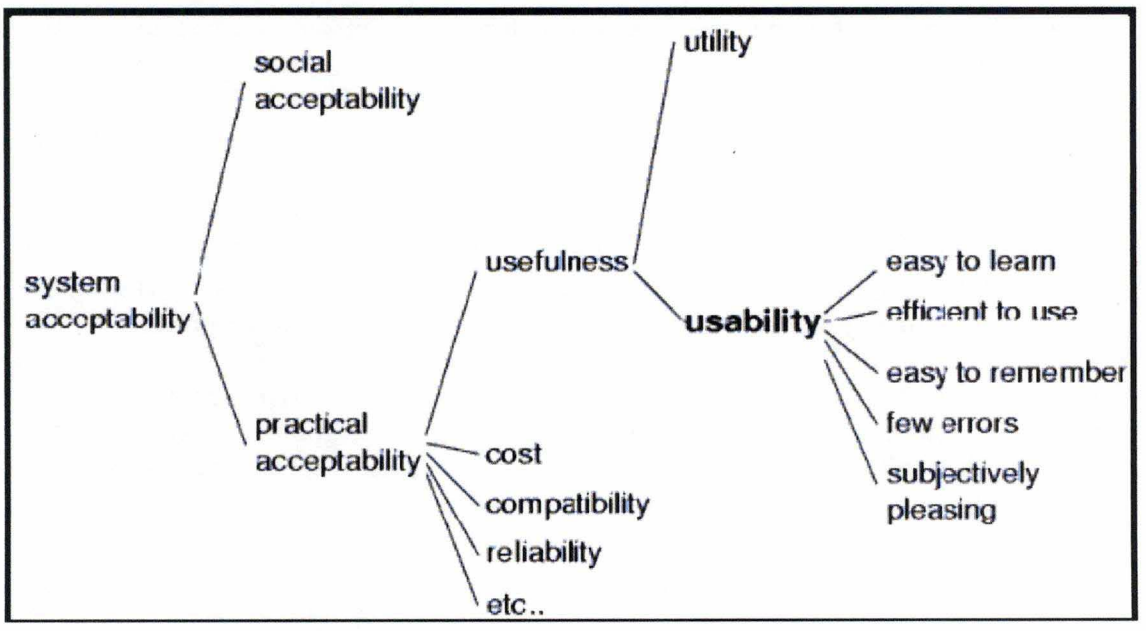

Figure 19: The place of *usefulness* in the overall acceptability of a system, from [Nielsen93].

III. Interactivity and Graphical Visualization

Since our senses have developed for thousands of years in a physical world, we explore and understand reality in physical terms. Geometric shape, dimension, position, colour, motion and other attributes are the signs of reality, which convey its information most efficiently. Our ability to perceive information in numerical or textual form is not as strong, and that is the reason why people find ways to analyse data virtually.

Visualization can be used to interactively manipulate the set of results and help the user find trends and relationships in the items more quickly than a traditional spreadsheet. This seems plausible given the presumed strengths of interactive 3D visualization techniques :

- Visualization helps users comprehend large quantities of data.
- Visual attributes can present abstract representations of data.
- Relationships among displayed entities become apparent.
- Graphical techniques allow more direct intuitive interactions with the entities of interest.

Furthermore interactive visualizations are :

- **dynamic,** letting the user interacts with multiple, linked views at once to see the same data from different perspectives.
- **interactive,** allowing the user brings in and changes additional dimensions through visual cues such as colour, size and position.
- and let the user makes queries by **directly manipulating** data in graphics : selecting, filtering, zooming, rotating. We can say the user almost simultaneously searches for data, retrieves them and interprets them.

The key to direct manipulation design is to create a visual representation of the "world of action" that includes selectable displays of the objects and actions of interest. Then with pointing, zooming, panning, and other interactions, the user can rapidly perform operations, see the results immediately and reverse operations if necessary.

[Shackel91] gives the plausible benefits of direct manipulation:

> "The benefits include relatively rapid learning with high retention over time and high user satisfaction. Errors can often be prevented because the representation shows the users the impossibility of performing a task and because typographic errors are eliminated when the user selects from a set of displayed objects. Exploration is often encouraged in direct manipulation environments, especially when reversibility of action is ensured."

To emphasize this notion of direct manipulation, Darville C. and Van Espen S. have drawn our attention to the role of a Visual User Interface (VUI):

> "The Visual User Interface is a method of direct manipulation enabling the user to take a more active role in the process of visualizing and investigating data. The most important difference between a traditional Graphical User Interface (GUI) and VUI is that the user interacts directly with the on-screen graphics and the data behind the graphics without having to work with traditional GUI controls such as pull-down menus, dialogs or buttons. Consequently, the main asset is that the on-screen graph, chart or data representation **is central to the user's focus.**
>
> Briefly, the representation "plays a part" of the interface, resulting in the commitment of fewer cognitive resources."

The advantages of the VUI are so much important that more and more applications use it. In fact, if you take advantage of the VUI, the user can actually interact with the data and explore them. VUI makes possible interactions such as panning, zooming, rotating, changing viewpoints and drilling down. Thanks to these interactions, you can more easily observe trends, relationships, distributions, anomalies, and so forth. The task is easier and faster. Most of these characteristics are available in the *Visualization Application* via the Cosmo Player[15] navigation tool. But an important dimension of this interactivity, concerning the modification of a chart directly in the virtual world, is missing to the *Visualization Application*.

[15] Cosmo Player is the *VRML* browser we will use for the application (cf. II.3 in the chapter four concerning the Implementation).

IV. Two-Dimensional Versus Three-Dimensional

According to Jakob Nielsen in [Nielsen98b], using 3D on a computer adds a range of difficulties :

- The screen and the mouse are both 2D devices, so we don't get true 3D unless we strap on weird head-gear and buy expensive bats (flying mice).
- It is difficult to control a 3D space with the interaction techniques that are currently in common use since they were designed for 2D manipulation (e.g., dragging, scrolling).
- Users need to pay attention to the navigation[16] of the 3D view in addition to the navigation of the underlying model: the extra controls for flying, zooming, etc. get in the way of the user's primary task.
- Poor screen resolution makes it impossible to render remote objects in sufficient detail to be recognizable; any text that is in the background is unreadable.
- The software needed for 3D is usually non-standard, crash-prone, and requires an extra download (which users don't want to wait for).

In particular, **navigation through a hyperspace** (such as a website) is often very confusing in 3D, and users frequently get lost. 3D navigation looks very cool in a demo, but that is because you are not flying through the hyperspace yourself. Therefore, you do not have to remember what is behind you or worry about what remote objects are hidden by near-by objects. The person giving the demo knows where everything is.

Most **abstract information spaces** work poorly in 3D because they are non-physical. If anything, they have at least a hundred dimensions, so visualizing an information space in 3D means throwing away 97 dimensions instead of 98: hardly a big enough improvement to justify the added interface complexity. This is Nielsen's standpoint but he considers here an extreme case, namely the hundred dimensions. For this case, it is obvious that 3D techniques will not improve the visualization. This is due to the fact that, in 2D, the visualization will already be poor and not representative.

In fact, a lot of Nielsen's arguments feel into thinking that the problem is not that 3D is inherently worse than 2D, but that we don't have good ways of using 3D interfaces yet. Therefore, according to him, more than really responding to a need, 3D and *Virtual Reality*[17] in Data Visualization are mainly used because they are in

---

[16] Navigation : movement and directing of movement through a space, such as a *virtual environment.*

[17] Virtual Reality, also called virtual environments, is a new interface paradigm that uses computers and human-computer interfaces to create the effect of a three-dimensional world in which the user interacts directly with virtual objects. [Bryson96]

fashion. The technology is not yet mature to be used. This approach of 3D visualization is quite severe. Yet the 3D has some real advantage over the 2D.

For instance, unlike Nielsen, we think that the fact to be able to visualize an additional dimension is a plus that we cannot neglect. A comparison between three-dimensional data is often more interesting than between two-dimensional data.

Moreover 3D may be particularly useful when you have to visualize a large number of items. The third dimension allows you to visualize more items. If you do not use it, you will have some complications in arranging the items. It will be difficult for the user to distinguish them visually if they are placed so they overlap or are very close to each other (in the ScatterChart for example). Of course, this might be remedied by detecting items collisions and highlighting them (using colour, transparency, etc.). But the dimension that would be used (colour or transparency) would be lost while it could still be used elsewhere in the virtual representation.

For an effective 3D visualization we must use an appropriate navigation tool and a appropriate way to represent the data (e.g. scaling the values of some variables). The depth in 3D allows a best cost-effectiveness of the screen space only in *Virtual Reality* charts[18] (VR charts refer to charts visualized in a virtual environment). Large data sets are totally unsuitable in 3D charts (3D charts refer to charts displayed by 3D visualization that is to say a visualization by means of static 3D charts on which none interaction is possible) where backwards data are inaccessible and in which the informational content is overloaded by the excess of information. Charts become subsequently unusable and difficult to interpret. It's then worse than 2D charts. At the opposite, interactivity provided by *Virtual Reality* allows manipulations of the graphs but also creations of various meaningful views from a single voluminous chart.

To illustrate the previous paragraph, we have generated (via the *Visualization Application*) a 3D BarChart. This chart conveys the average in *Arithmetic*[19] (y-axis) for the students of the University of Port Elizabeth, with regard to their *Race* (x-axis) and their *Faculty* (z-axis). 777 students are represented on this chart. The next figure (Figure 20) shows a static view of the chart generated. Of course, as this chart is embedded in a virtual environment, the user would be able to navigate throughout the world. But not if it was only a 3D chart (that is to say, not displayed in a virtual world).

---

[18] The definition and the comparison of VR charts and 3D charts has been fulfilled by Darville C. and Van Espen S., in "*Introduction to Semantic Properties of Business Data*" [Darville&2000].

[19] *Arithmetic*, *Race* and *Faculty* are measurable characteristics of the students of UPE. These characteristics are available in the database UPE kindly gave us. In consideration of protecting the individual privacy, the names and surnames of the students were removed.

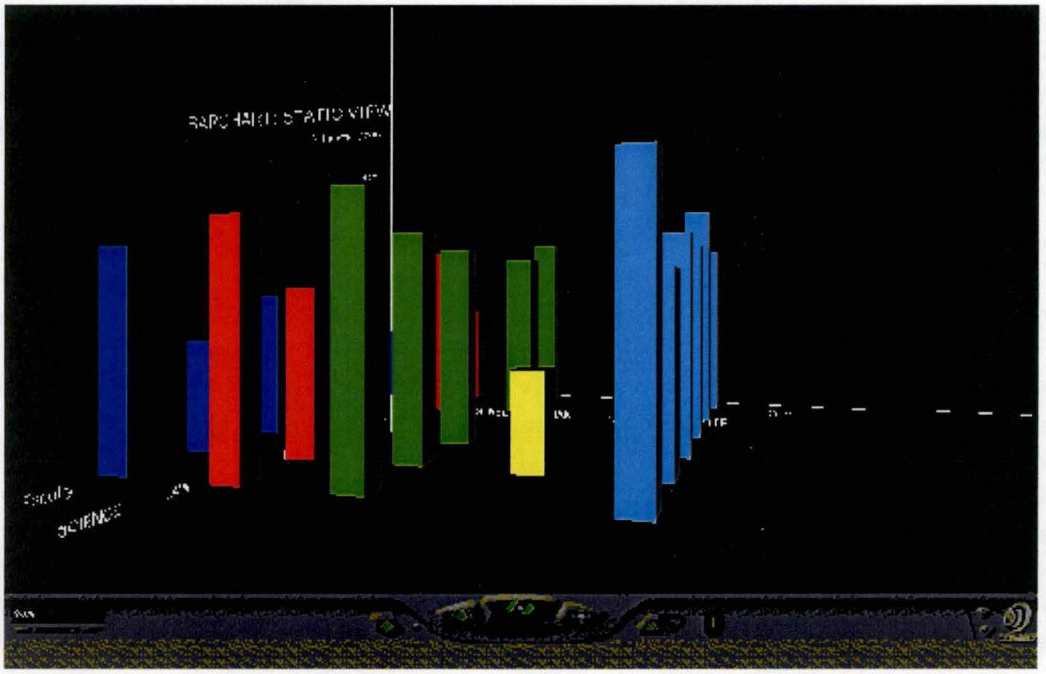

Figure 20: Static view of a 3D BarChart.

In fact, as this figure represents only a static view of the produced BarChart, we can consider it as a 3D chart, in contrast with VR chart. So, we can show easily that the backwards data are inaccessible, and consequently the chart seems to be overloaded by the excess of information. Of course, in order to have access to the backwards data, we could have chosen a better static view of the VR chart (for example, a view from the top). But in this last case, the user would not be able any more to distinguish the height of all the different bars.

In conclusion, thanks to the *Virtual Environment*, in which the chart is built, the user is able to produce as many static views of the chart as he wants. No more data are inaccessible and the chart does not seem overloaded any longer. Because the user is the one who chooses the viewpoints from which he wants to analyse the chart.

V. Virtual Environments

In the previous paragraph we have introduced the concept of *Virtual Reality* charts. These charts are visualized in a *Virtual Environment* (VE). Now, we will present what the current literature thinks about the VEs, the user interaction in these environments and usability requirements to support that interaction.

VEs are also referred to as *virtual worlds*, and the concept they capture can be referred to as *virtual reality*. There are no generally agreed definitions of these terms, but the goal of VR systems is to place the user in a three-dimensional environment that can be directly manipulated. Ideally, users cease to think of themselves as interacting with a computer and interact instead with the 3D environment.

Loeffler and Anderson (1994) define virtual reality as an artefact:

> "Virtual Reality is a three-dimensional, computer-generated, simulated environment that is rendered in real time according to the behaviour of the user." [Loeffler&1994]

Ellis provides a more detailed definition of the artefact:

> "A virtual environment consists of content (objects and actors), geometry and dynamics, with an egocentric frame of reference, including perception of objects in depth, and giving rise to the normal ocular, auditory, vestibular, and other sensory cues and consequences." [Ellis1993]

Alternatively, Gigante defines the experience of interacting with a VE:

> "Virtual reality is an immersive, multi-sensory experience. It is characterized by the illusion of participation in a synthetic environment rather than external observation of such an environment." [Gigante1993]

Randy Pausch introduces also the notion of *Virtual Reality*, and compares it to motion pictures:

> "Virtual reality presents a synthetically generated environment to the user through visual, auditory, and other stimuli. Of course, motion pictures also attempt to place the user in synthetically generated environments. The two major differences between virtual reality and motion pictures are that virtual reality technology can create a much stronger illusion and that VR is an interactive, not passive, experience". [Pausch1993]

From these definitions, the key features that characterise VEs appear to include three-dimensional graphics and an environment model, representing some real-life or artificial structure or place. Components in a VE include a background space, the user, objects and possible agents. These environments differ from conventional interface types, bringing new challenges to human-computer interface design. In the next paragraph, we will compare the interface structure of VEs with a predecessor, Direct Manipulation.

The comparison between VEs and Direct Manipulation[20] (DM) interfaces highlights some major differences:

- VEs are structured as 3D graphical models. Therefore, they involve an additional dimension and consist of graphical objects embedded in a world, as opposed to a mixture of graphical icons and text labels in DM interfaces.

- The VE model represents some real-life or artificial structure or place, that has a fairly static spatial organisation. DM systems provide a 2D display area that continually presents objects of interest to the current task.

- Only a sub-section of the model is available through the VE interface at any one time, according to the current user position. Therefore, the user has a limited view and must navigate around the model to locate objects of interest and interact with the environment. There is no concept of user position in DM interfaces (apart from the mouse cursor). The user manipulates the DM interface as required, for example by moving or re-sizing interfaces objects and opening/closing windows.

Direct Manipulation interfaces have promoted interaction based on the user's manipulation of computer-based objects, on the grounds that this would place less load on the human cognitive system and is preferred by users. DM is characterized by the following properties:

- continuous representation of objects of interest,

- physical actions or labelled button presses instead of complex syntax, and,

- rapid incremental reversible operations whose impact on the object is immediately visible.

The notion of directness is related to the psychological distance between user goals and user actions at the interface, and the psychological engagement of feeling oneself to be controlling the computer directly rather than through some hidden intermediary.

DM interfaces have an established paradigm of interaction (WIMP – Windows, Icons, Mouse, Pop-up menu) but VEs have yet to evolve a dominant interaction paradigm. VEs reduce psychological distance by representing task domains in a more realistic manner and facilitating more natural, multi-modal interaction [Bryson95]. The computer is removed as an object of perception, allowing the user to interact directly with the generated environment [Hubbold&93].

---

[20] Direct Manipulation is an interface style that promotes *interaction* based on the manipulation of continually represented computer-based objects, such as icons, menus and windows.

The following figures illustrate those differences. Figures 21 and 22 give screenshots of example DM interfaces, showing 2D display areas for current interface objects of interest, such as icons and menus. Figure 23 and 24 give screenshots of example VE interfaces, showing viewpoints to 3D large-scale structures. But watch out! These two last figures represent only a static view of the 3D structures.

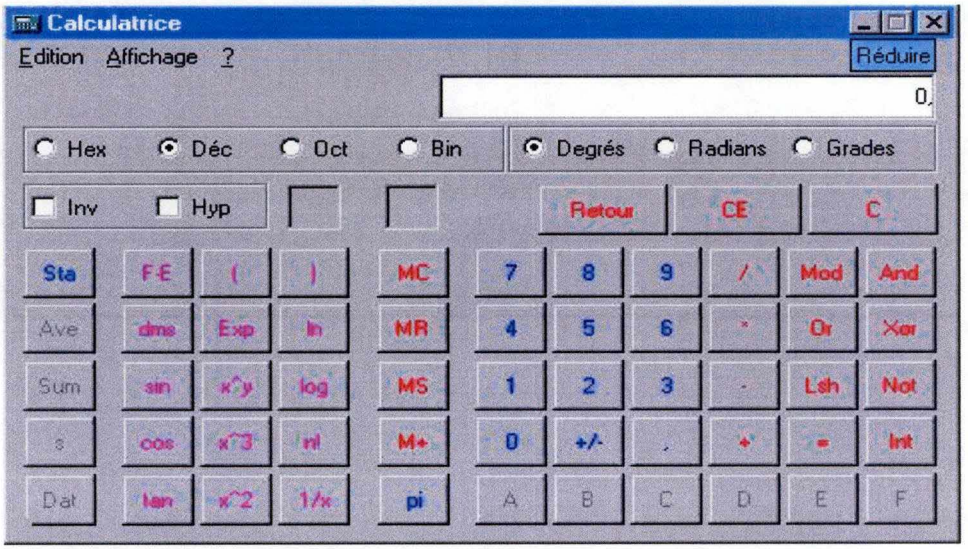

Figure 21: Screenshot of the Windows Calculator DM interface.

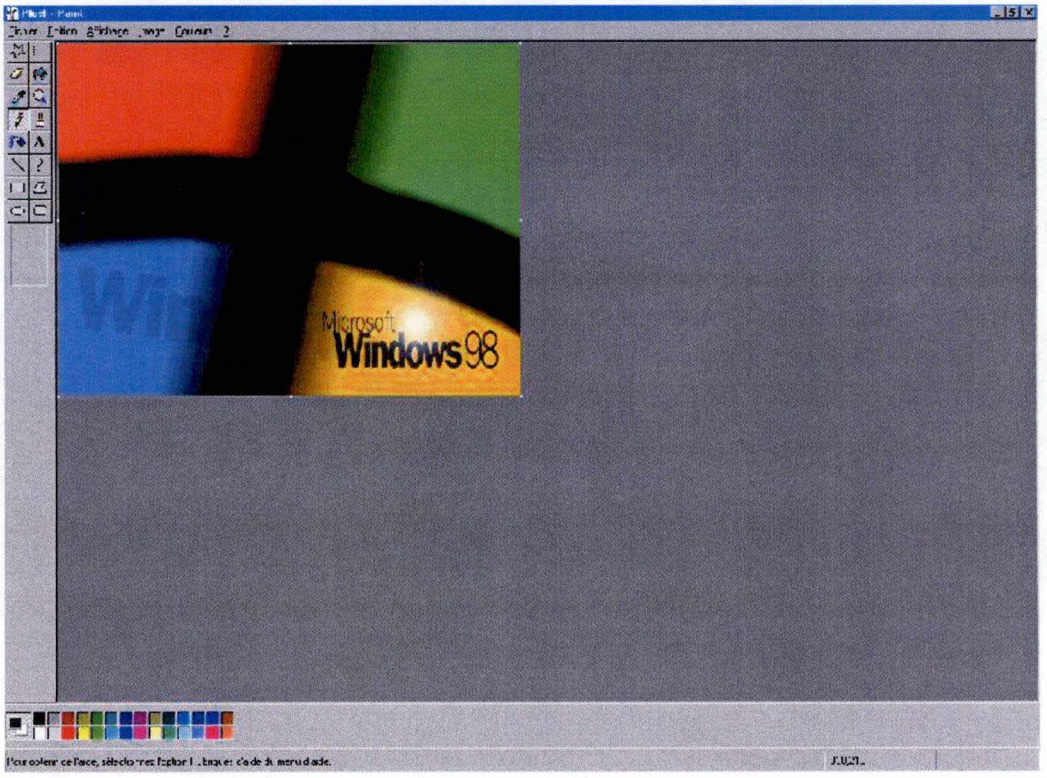

Figure 22: Screenshot of the Paint DM interface.

Figure 23: Screenshot of the Detroit Midfield Terminal VE interface, developed by the *Virtual Reality Laboratory* (VRL) at the College of Engineering at the University of Michigan for the Detroit Metropolitan Airport (from http://www-vrl.umich.edu/NewMidfield/index.html).

Figure 24: Screenshot of a Demo Game VE interface.

Objects interaction in VEs share some similar features to those in DM systems. Objects are manipulated directly, for example by picking up and moving them, although in VEs the third dimension may also be involved, for example grasping

and rotating objects. Furthermore, some toolkits include DM facilities such as buttons and menus, as well as common world components such as walls, lights and rooms. Therefore, VEs continue from and have some similar qualities to DM systems.

Other special features particularly include the more exploratory nature of interaction in VEs, due to the typical open-ended task structure. Virtual environments are often active, with objects operating independently of the user's actions. They typically model real world domains, and use multi-modal and natural interaction styles, such as human-like navigation and viewpoint control. A general aim of VEs is to promote user presence, the experience of "being in" or, with a lesser degree, "being in contact with" the VE [Loomis93].

Where real-world phenomena is modelled, VEs may be especially useful if the physical counterpart cannot be visited for reasons such as its size, location or danger. Useful applications of *Virtual Reality* include training in a variety of areas (military, medical, equipment operation, etc.), education, design evaluation (virtual prototyping), architectural walk-through, human factors and ergonomic studies, assistance for the handicapped, entertainment, and much more [Beier2000]. At the other hand, VEs are not useful for applications where it is not appropriate for the user to move around the space and experience it from different positions.

Another interesting field of application for the VR concerns the study and the treatment of phobias by virtual exposure (e.g., fear of heights, fear of flying, agoraphobia, fear of spiders, fear of driving and so forth). Specific virtual environments are designed to aid in mental rehabilitation and to assist patients in overcoming phobias. The main advantage is that the computer is more controllable and inexpensive than the conventional in vivo treatment technique, and most researches are now using PCs in their labs to create the phobia images [Strickland&97].

However, the novel concepts and technology involved with VEs lead them to be significantly more difficult to design, implement and use than conventional interfaces. For example, some authors suggest adaptive rendering techniques to maintain performance, such as only showing the parts of the graphical model within the field of view. Besides, users interacting with VEs have been known to suffer from frequent difficulties, which can result in frustration and an overall low system usability.

Common problems are disorientation, perceptual misjudgements and confusion with unnatural interactions. In part, this is because although, in an ideal VE, interaction would perfectly mimic interaction in the real world, in practice, VEs often differ from their real world counterparts. There may be necessary limitations in interactions, such as the absence of tactile feedback, substitutions, such as the use of gestures for navigation instead of whole body movement, or empowerments, such as the ability to walk through walls. The user needs to be able to adapt to and tolerate limitations, understand and adapt to substitutions, and take advantage of empowerments.

Next, virtual environments can be differentiated according to their relationship to the real world and the hardware involved:

- *Immersive environments* involve immersive head-mounted displays and do not include data from physical reality.

- *Desktop* (or *non-immersive*) *environments* involve desktop displays, which allow the user to maintain awareness of physical reality.

Immersive environments have the potential to provide the user with an absorbing experience of the environment. However, they can be isolating, involve intrusive technology and have been associated with health problems [Travis&94]. Desktop environments avoid these problems, but do not provide a fully immersive experience, instead relying on psychological immersion taking place [Robertson&93].

So we can obviously say that the *Visualization Application* takes place in a desktop environment. In fact, the VRML scenes displayed do not involve any immersion. But the scenes are displayed with the same 3D depth cues used in immersive VR: perspective view, hidden-surface elimination, colour, texture, lighting, shading, and shadows. Besides, the users of the application will probably be unwilling to put on special equipment to do their work, especially if it obscures their surroundings. Non-immersive VR does not prevent users from seeing what is around them and does not require wearing any special equipment.

> "The use of immersive VR for extended time periods is likely to cause psychological and physical stress that most users will not tolerate. Non-immersive VR stress factors are the same as those for general computer use and are likely to be much less than for immersive techniques ". [Robertson&93]

Both styles offer to the user basic interactions in a VE. These user interactions are:

- *navigation* and *viewpoint control*, and

- *object interactions*, such as picking an object; grabbing, rotating and moving objects; manipulating objects to change their state; and, querying to find out the content of an object.

Providing support for user interaction is important and various techniques have been investigated to support different interaction tasks. Support can include guiding the user during exploration to ensure they are not lost, bored or overloaded with information. To aid object interaction, visible object-handles can be provided using 3D geometry to better enable manipulation.

General movements tasks that users carry out include orientation, navigation, way finding and exploration. Three way findings tasks are *naïve searches*, where there is no knowledge of the target whereabouts, *primed searches* where the target location is known, and *exploration* where there is no relevant target.

General object tasks include inspecting and interacting with objects. When the user wants to inspect or interact with an object they must navigate and approach the object, controlling their rotation and velocity so as to end up in the desired orientation and position. The desired orientation and position is one at which the user can discern relevant visual detail or manipulate the structure. [Rushton&93]

Users have limited perceptual awareness and focus when interacting, depending on their current position in the environment. Users need to build up an understanding of their embodiment in the VE and its capabilities, as well as an understanding of the world and its spatial structure. In an ideal interaction with a VE, the user would feel present in and directly engaged with the environment. However, users can suffer from common interactions problems.

Indeed, during the interactions with the VE the user can cope with different problems. So, maintaining general spatial orientation can be difficult. Disorientation and losing whereabouts commonly occur when the navigation speed is too fast or the movement direction is not as expected. Perceptual problems can be common, such as difficulties distinguishing objects, and difficulties in size and depth perception. There can be incorrect perception of movement with users feeling objects are moving towards them instead of feeling they are flying through the world. Specific technical problems, such as slow display update rates, can also result in usability issues, as well as health problems such as motion sickness.

Misjudgements of depth and distance may be due to the fact that the eyes focus less accurately on virtual, rather than real, images [Carr&93], or because of the lack of anchor points (e.g. a ground plane) or variations in some virtual images [Loomis93]. Motion sickness is typically caused by continuous, unexpected or unfamiliar, sensory information concerning the orientation and movement of the body. However, few of these suggestions have been translated into specific design guidance for avoiding the problems that exist.

VI. Conclusion

First of all, the use of *Virtual Reality* for the Interactive Visualization has no negligible benefits. Besides those benefits lessen the negative impact of three-dimensional visualization. However, interactivity and manipulations demand an outstanding interface in order to be usable. Indeed, some interactions such as flying or rotating can cause disorientation to the user and confuse him, with the negative consequence of reducing the usability of the Visual User Interface.

Three-dimensional charts in virtual worlds are adequate for depicting the general nature of data and are almost never used to convey specific values, since it is difficult to determine exact values from such charts. But the visualization tool should provide the user with the possibility of getting knowledge of such values.

The interactivity, the best cost-effectiveness of the screen space, the various visual indications and the Visual User Interface are the key factors of the Interactive Visualization, allowed by the *Virtual Reality*. These factors give the *Virtual Reality* a power which is able to convey information very efficiently as well as to provide interactive capabilities to exploit the maximum of data. Unfortunately, we could not exploit all of these capabilities in order to provide the user with a full interactivity (as mentioned in the *Introduction*).

The promise of VR is beginning to be recognized. Interesting, useful applications of VR are emerging in areas as diverse as medicine, design, the arts, scientific visualization, education, and entertainment. Nevertheless, the development of VR systems is often a time-consuming, difficult, and frustrating process [Singh&1996]. The major concern is the fact that these means of visualization are new in the field of data analysis and many researches have to be led in order to settle human visual perception problems and, to propose concrete and precise guidelines for the development of usable interfaces.

Chapter Three

Design of the *Visualization Application*

I. Introduction

The purpose of this chapter is initially to give the specifications and the architecture for the part of the *Visualization Application* concerning the generation of different charts, and to propose a prototype for the User Interface (UI).

We have first specified an architecture that should be general for any chart. Then, by using the principles of inheritance, we can propose a specific architecture for a given chart.

The first chart we have worked on is the *ScatterChart*. We have chosen this one because of its ability to include so many different dimensions. In addition, as we consider big data sets, it has to accommodate large amounts of data. And the *ScatterChart* does this. For example, if one wants to identify trends and exceptions, the chart should display plenty of information.

Similarly we were able to build the architecture for the *3D BarChart*. And as we will see later, the implementation is quite different. But both implementations extend and use the implementation provided by the generic chart class. Then, we will propose an interactions' scheme that features and explains the different interactions between the main components (only the ones that are directly related to the building of virtual charts) of the architecture. At the same moment, you will be able to see an example of how this architecture can be extended to include other charts.

Afterwards, we are going to expose what we've done concerning the conception of the User Interface and will discuss the design of this interface.

And eventually, we will summarize all this by putting forward a global architecture of the *Visualization Application* in its entirety, including the interface components, the database accesses, and so on.

II. Technical specification of the generic Chart

This architecture will highlight what we need to know to correctly build the code and will be a guideline for the implementation. As our implementation language is *Java*[21], the terminology used here comes from this programming language.

We will follow the *Basic UML* notation for classes as referred in [Wessonb2000] and as illustrated in Figure 25. This *Basic UML* notation takes over only a subset of the *UML* notation and complies well with our aim of giving only an overview of the classes used in the global architecture.

---

[21] This choice will be justified in the chapter four *Implementation* (Section II. **Implementation tools**).

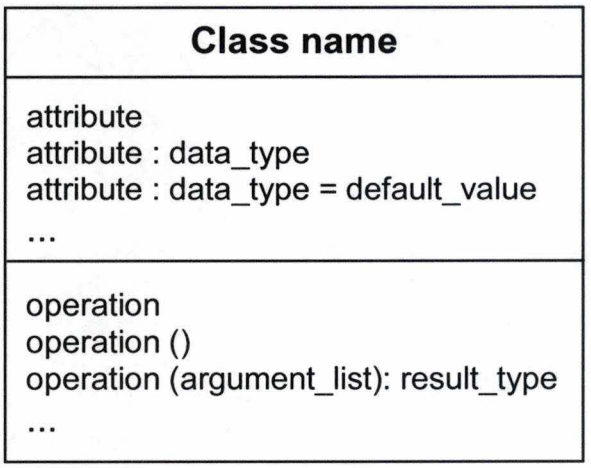

Figure 25: *Basic UML* notation for classes

A Java class in our implementation will effectively represent each of these classes. This design gives an overview and consequently the classes here do not contain everything that the Java classes will contain.

The articulation between all those classes will be given in the section IV. Before, we will present in the section III the technical specifications for the two charts we manipulate in the current *Visualization Application*, that is the *ScatterChart* and the *BarChart*.

II.1 The Chart class

Chart (abstract class)	
semantics: Vector of Semantic[II.3]	(1)
title: String	(2)
browser: Browser	(3)
root: Node	(4)
addRow (Datum[II.2] data) *(abstract method)*	(5)
Chart (String title, Vector semantic) *(constructor)*	(6)
set_browser (Browser b)	(7)
set_ancetre (Node n)	(8)

Figure 26: The Chart class

By using a generic class and the inheritance principles, we provide the potential future designers with a certain flexibility concerning the possible charts they want to manipulate.

The *Chart* class, as we will see later, will be declared as an **abstract**[22] Java class. In fact, one of its methods (*addRow*) contains no implementation. This is due to the fact that this method is particular to each specific chart. The remainder is identical: so the building of the legend in the VRML world is obviously the same for any chart. The constructor class contains the part that will allow the creation of the legend.

A discussion of this class is given below:

(1) Each instance in this vector[23] will describe the semantics of each selected column, related to a dimension of the data. See also the *Semantic* class (paragraph II.3).

(2) We can also have a title for the chart.

(3) This is the browser that will display the VRML world containing the *Chart*. **Browser** is a VRML Object.

(4) This object represents a basic VRML world to which we will add the chart. **Node** also is a VRML Object.

(5) This method adds a specified row to the chart. The data parameter represents this row. This is an **abstract** method and so it contains no implementation.

(6) The constructor needs a String title and a Vector of Semantic Objects (containing the semantic of each column). This constructor will initialise (mostly the legend on the axes) the VRML world so that it will only be necessary to add a point at a time to the world.

(7) & (8) The main class (the *Java Applet* that constitutes the starting point of the Graphical User Interface) used to build the chart has to initialise the variables browser and node, via these both methods.

---

[22] In Object-Oriented programming, there are cases in which it is useful to define classes for which the programmer never intends to instantiate any objects. Such classes are called *abstract* classes. They are used as superclasses in inheritance situations. The sole purpose of an abstract class is to provide an appropriate superclass from which other classes may inherit interface and/or implementation. Classes from which objects can be instantiated are called *concrete* classes.

[23] A vector represents a data structure (such as a list) that contains a variable number of objects.

II.2 The Datum class

This class manages the data from the database and provides a chart class (e.g. the *ScatterChart* or the *BarChart* class) with the required data. This class does not access the database. It gets the current row to add to the chart and inserts it into the values Vector.

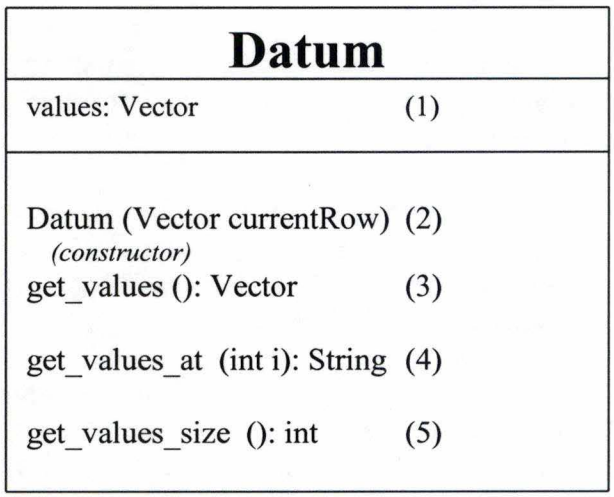

Figure 27: The Datum class

A discussion of this class follows:

(1) Here we can find the values (strings) required from a class that instantiates the *Datum* class. We will use these values to build the chart.

(2) This constructor will require a vector representing the current row of data.

(3) This method returns the values vector.

(4) This method returns an element of the vector at a specified position

(5) This method returns the size of the vector.

II.3 The Semantic class

This class describes the semantic of a column. It includes the name of the column, the type of the column (ordinal, nominal or quantitative), a legend and an instance of the *Representation* class. This last class will manage the representation of a specific column in the chart (see II.4).

Semantic	
name: String	(1)
type: String	(2)
legend: String	(3)
rep: Representation[II.4]	(4)
Semantic (String name, String type, String legend, Representation rep) *(constructor)*	(5)
get_name (): String	(6)
get_type (): String	(7)
get_legend (): String	(8)
get_rep (): Representation	(9)

Figure 28: The Semantic class

A discussion of this class is given below:

(1) The name of the column (e.g. Gender).

(2) The type of the column (for Gender we will have the **nominal** type).

(3) The legend one wants to give for the association.

(4) This associates a dimension with the column. But for more details see the Representation class (paragraph II.4).

(5) The constructor needs four parameters: a name, a type, a legend and an instance of the Representation class.

(6) The different get methods will return either the name field, or

(7) The type field, or

(8) The legend field, or

(9) The Representation instance of the current Semantic instance.

II.4 The Representation class

As discussed earlier this class describes how we want to visualize a specific column. So we need the dimension (a string) that we want to associate with the column. If the dimension is shape, colour, transparency or texture we need a "function" that associates with each possible value for the column a specific shape, colour, texture or a transparency value.

Instead of a function we can use two arrays (same size) of strings. One array will contain all the possible values for a column, and the other one will contain the different colours (shapes, textures or transparency values) that we have chosen for the association.

For the quantitative dimensions, one array will contain the actual values and the other one will contain the VRML values. For example, if the dimension is the x-axis, the first array, *datavalues*, can be [0,100] and the other one, *dimvalues*, could be [0,10], where 100 is the maximum actual value and 10 is the maximum value in the virtual world. For nominal or ordinal data that we want to plot on a particular axis, *datavalues* will contain all the possible values and *dimvalues* will contain their position in the VRML world with regard to the dimension field.

Currently we have to propose to the user all the possible shapes that we can handle easily in VRML (Box, Cylinder, Cone and Sphere are the predefined shapes). Other 3D shapes are possible but it would be a difficult task for the user to specify everything that VRML has to know. Otherwise, if necessary, we could widen the set of predefined shapes.

For the texture, the user should provide the files to be used. But we can also supply a predefined set of textures. Texture image files must be in the *JPEG*, *GIF*, or *PNG* image file formats.

Representation	
dimension: String	(1)
datavalues: String []	(2)
dimvalues: String []	(3)
Representation (String dimension, String datavalues [], String dimvalues []) *(constructor)*	(4)
get_dimension (): String	(5)
get_datavalues_at (int i): String	(6)
get_dimvalues_at (int i): String	(7)

Figure 29: The Representation class

A discussion of the previous class follows:

(1) The dimension value (colour, shape, size, texture, transparency, x-axis, y-axis, or z-axis). But for the *BarChart*, the only possible values for dimension are the three axes.

(2) The different values for a column (e.g. [male, female] for the gender column).

(3) The different values for the dimension specified in the dimension field. For example, if we've chosen colour to represent the gender column, and we want to visualize male as blue and female as pink, the array will be [blue, pink].

(4) We need a constructor that takes three parameters (the dimension and the two arrays).

(5) The different get methods will return either the dimension, or

(6) The element at position i of the datavalues array, or

(7) The element at position i of the dimvalues array.

III. Technical specification of the specific Charts

III.1 The ScatterChart class

ScatterChart	
semantics: Vector of Semantic[II.3]	(1)
title: String	(2)
browser: Browser	(3)
root: Node	(4)
addRow (Datum[II.2] data)	(5)
ScatterChart (String title, Vector semantic) *(constructor)*	(6)

Figure 30: The *ScatterChart* class

A discussion of this class is given below:

(1) – **(4)** Cf. paragraph II.1. In fact, as *ScatterChart* **extends**[24] the *Chart* class, it will also inherit from the *Chart's* declaration of variables. So, in the implementation of the *ScatterChart* class, we will not define these variables again.

(5) This method adds a specified row to the chart. The data parameter represents this row. It contains an implementation specific to the *ScatterChart*.

(6) The constructor needs a String title and a Vector of Semantic Objects (containing the semantics of each column). In this constructor, we just need to call the constructor of the *Chart* class, with the right arguments. This will initialise (mostly the legend of the axes) the VRML world so that we will only have to add ***a point*** at a time to the world.

[24] Inheritance is a form of software reusability in which new classes are created from existing classes by absorbing their attributes and behaviours and embellishing these with capabilities the new classes require [Deitel&1999]. A class inherits from another one via the keyword **extends**. The first one is referred as the subclass of the second, and the second one as the superclass of the first.

III.2 The *BarChart* class

BarChart	
semantics: Vector of Semantic[II.3]	(1)
title: String	(2)
browser: Browser	(3)
root: Node	(4)
addRow (Datum[II.2] data)	(5)
BarChart (String title, Vector semantic) *(constructor)*	(6)

Figure 31: The *BarChart* class

(1) – **(4)** Cf. paragraph II.1. In fact, as *BarChart* **extends** the *Chart* class, it will also inherit from the *Chart's* declaration of variables. So, in the implementation of the *BarChart* class, we will not define these variables again.

(5) This method adds a specified row to the chart. The data parameter represents this row. It contains an implementation specific to the *BarChart*.

(6) Our constructor needs a String title and a Vector of Semantic Objects (containing the semantics of each column). In this constructor, we just need to call the constructor of the *Chart* class, with the right arguments. This will initialise (mostly the legend of the axes) the VRML world so that we will only have to add ***a bar*** at a time to the world.

III.3 Other charts

As we have seen it in the paragraphs III.1 and III.2, the specifications of the *ScatterChart* and the *BarChart* classes are similar. They both extend the *Chart* class, and the only non negligible difference is the implementation of the *addRow* method.

So in order to be able to add other charts to the application, there is just a need to give a specific implementation of the *addRow* method of the new class. This implementation should comply with the semantic properties of the looked-for chart. That is almost all. In fact, the designer will have also to modify the User Interface in order to manage the specificities of the new chart, such as the possible dimensions and their number, the constraints of representation, and so forth. An example will be given later in this chapter.

IV. Interaction between classes

IV.1 Class Diagram

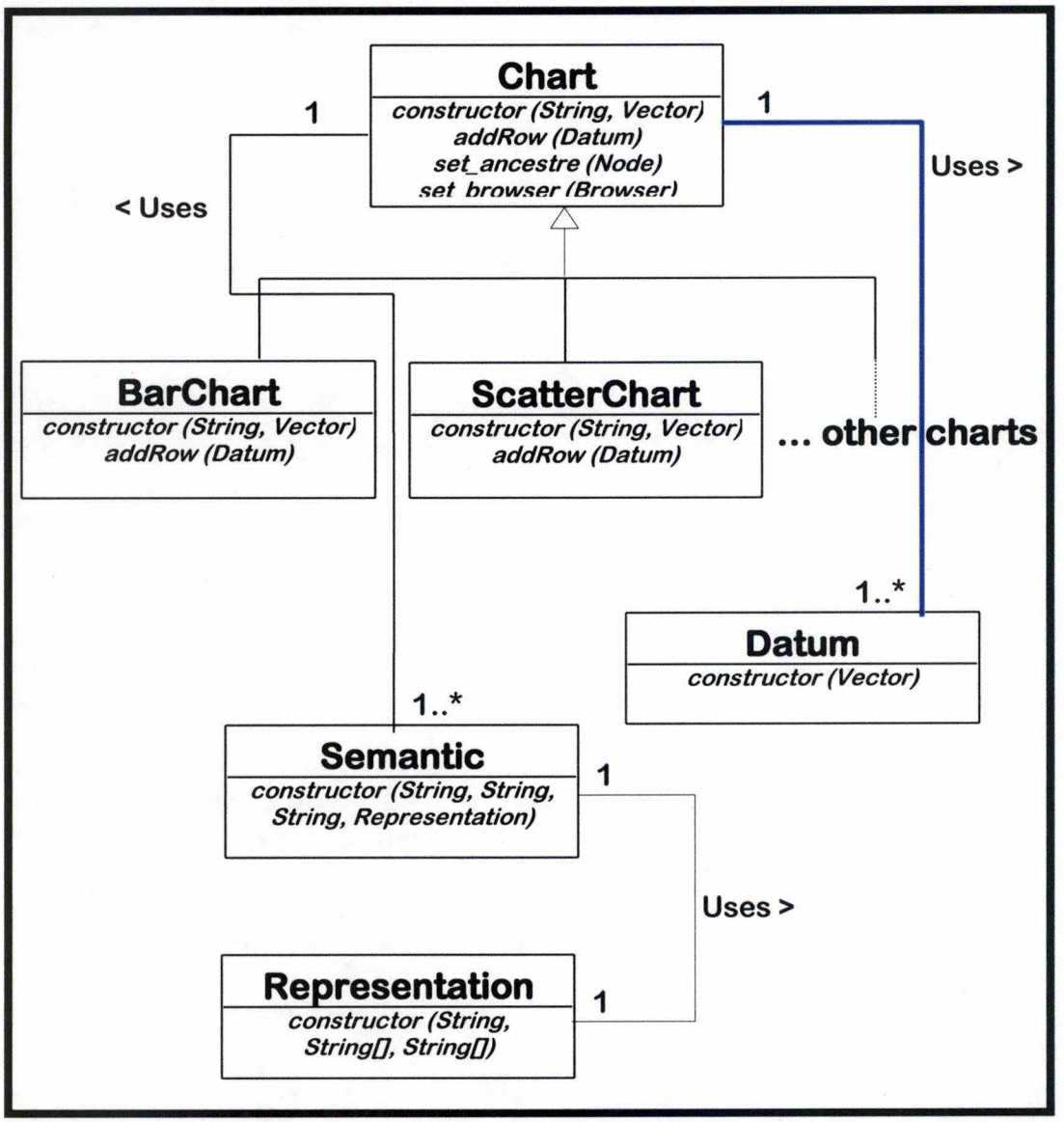

Figure 32: General Architecture

We use here the *UML* notations. The word **"Uses"** is simply the name of the association between the two classes. The ">" or "<" symbol indicates the sense of lecture (e.g. **Chart "Uses >" Datum** means that the *Chart* class uses the *Datum* class). The cardinalities of the associations are indicated close to the classes. Their convention is the opposite of the *ERA*[25] model. The cardinalities on the thick arrow (in blue) between the **Chart** and the **Datum** class mean that the **Chart** class uses at least one **Datum** class and that this last one is used by one and only one **Chart** class.

[25] ERA is an acronym for Entities-Relationships-Attributes Model.

As we can see it in this figure, the *ScatterChart*, the *BarChart* and the other possible charts **extend** the *Chart* class (the link between those classes is materialized by an empty arrow, like this one —▷). This means that they inherit the content of the **superclass**, i.e. the *Chart* class. Besides, as *Chart* is an **abstract** class and contains an **abstract** method, the classes that extend *Chart* must give an implementation to this method. Otherwise, these classes should also be declared as **abstract**.

The other relationships with the *Chart* class bond this one to the *Datum* class and to the *Semantic* class. They are **"Uses"** relationships. This means that the *Chart* class can access and instantiate the other classes. Doing so, it will be possible to call the **public**[26] methods and to access the **public** fields of the instantiated class.

As *ScatterChart* and *BarChart* **extend** the *Chart* class, the link between the first ones and the *Datum* and *Semantic* classes is the same, i.e. a **"Uses"** relationship. We have not represented these interactions on Figure 32 because it is implicit, by the nature of the **"extends"** relationship.

The *Datum* class is used to convey the data a *Chart* class (*ScatterChart* or *BarChart*) will have to represent as a VRML object. The *Semantic* class is used to convey the semantic properties of each elementary type of data. For example, in our application, an instance of the *Datum* class will contain all the values for a particular student with regard to some selected characteristics that seem pertinent to the user (e.g. gender, race, faculty, average, and others). So, there will be a need to have one and only one *Semantic* instance for each of these characteristics.

In a similar way, the *Semantic* class **"Uses"** the *Representation* class. It seems obvious if we take a closer look at Figure 32: the *Semantic* constructor needs an argument that represents a reference to an instance of the *Representation* class. This instance is used to express the relationships between actual values of a data, and values required for the virtual representation (cf. II.4 for more details).

IV.2 Extension and Reusability of the *Visualization Application*

Even if we have abandoned the use of *Java Beans* technology, nevertheless our application is fairly reusable. A programmer could reuse the package that is composed of the described classes in the Figure 32. Of course, this solution is not as flexible as the one we had considered at the beginning via the use of the Beans components. The programmer won't be able to base his judgement on a graphical development environment in order to reuse our architecture without knowing the details of the classes and the relationships between them. He will have to start from the code of this classes, to know their functionalities and modify what is needed for the creation of the new chart, notably in the class that materializes this chart (i.e. a class having the same structure as the *ScatterChart* class). So, if he wants to visualize some data in the VRML world thanks to the new chart, he will be able to

[26] The keyword **public** in Java is a *member access modifier* [Deitel&1999]. Instance variables or methods declared with member access modifier **public** are accessible wherever the program has a reference to the object that contains these methods and/or variables.

do it on condition that he takes into account the whole architecture, except for the classes that are particular to charts in which he is not interested.

In a sense, we have built an architecture that is general for any chart. By using the principles of inheritance, we can easily extend the *Chart* class that is common to each chart, and implement a given chart. The implementation can be quite different between charts. But each one will extend and use the implementation provided by the generic chart class. By proceeding like this, we provide the potential future designers with a certain flexibility concerning the possible charts they want to manipulate.

For example, if we want that the *Visualisation Application* allows also the visualization of *Ribbon Charts*, we will have to add to the architecture a *RibbonChart* Class. This class would enable to display a *Ribbon Chart*. In [Darville&2000], they define this chart as follows:

> "The Ribbon Chart displays continuous quantitative information by means of thick lines, visually similar to ribbons. Typically, it has a quantitative scale attached to the vertical axis and a qualitative or quantitative scale on the other two axes depending on the data nature".

This chart is particularly useful to underline the data trend along a time period and highlight the evolution of data. The figure 33 illustrates that.

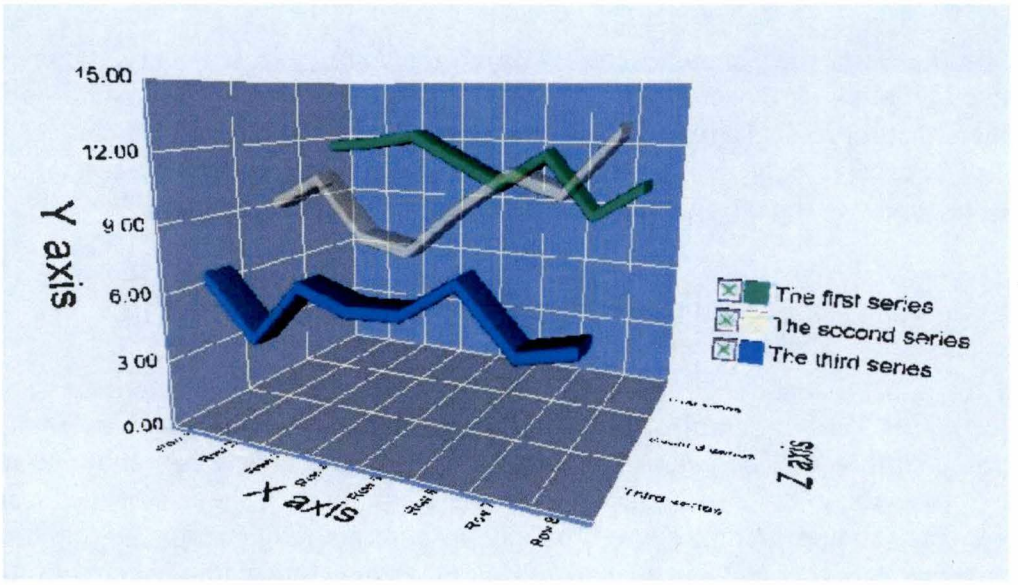

Figure 33: A Ribbon Chart

This chart looks pretty like the *3D BarChart*. There are several bars on the x-axis for the *BarChart* (cf. Figure 34a) whereas there is only one ribbon on the *Ribbon Chart*. This ribbon represents as much information as all the bars.

So to implement the new *RibbonChart* class, one will have only to write a new *addRow* method based on the *BarChart* implementation. But here a ribbon represents several rows, and not only one as in the 3D BarChart. So a new method is necessary, and its role is to collect all the rows that form a ribbon, build this ribbon as a VRML object, and then display it. The *addRow* method will be quite similar to the one used in the *BarChart*, except that it will no more add a VRML object directly to the virtual world.

The other main change concerns the building of VRML objects. In fact, here we have to display ribbons, and it does not correspond to a predefined shape (like the box, the sphere, the cylinder and the cone). The developer will have to put right this problem, but the solution can be very simple: for each ribbon, create several boxes that he will stick together. For instance, he will have to create seven flat boxes, in order to display a ribbon of figure 33.

V. Prototype of the interface

The User Interface has been written in Java, and mainly via the *Swing*[27] Components. This section's aim is just to present the general aspect of the User Interface, and not to explain the choices we have made. This topic will be discussed in the next section concerning the design of this interface.

The interface contains 5 frames. The first frame would make it possible to choose which chart one wants to use to visualize the data. The user would have the choice between the *3D BarChart*, and the *3D ScatterChart*, as we can observe it on the next figure. In this frame, we have added examples of representation of the two charts. So the user can choose the right chart he wants to manipulate.

[27] The Swing components are the newest Graphical User Interface components of the Java 2 platform. They are written, manipulated and displayed completely in Java (so-called *pure Java components*). So they provide a greater level of portability and flexibility than the original AWT components. [Deitel&1999]

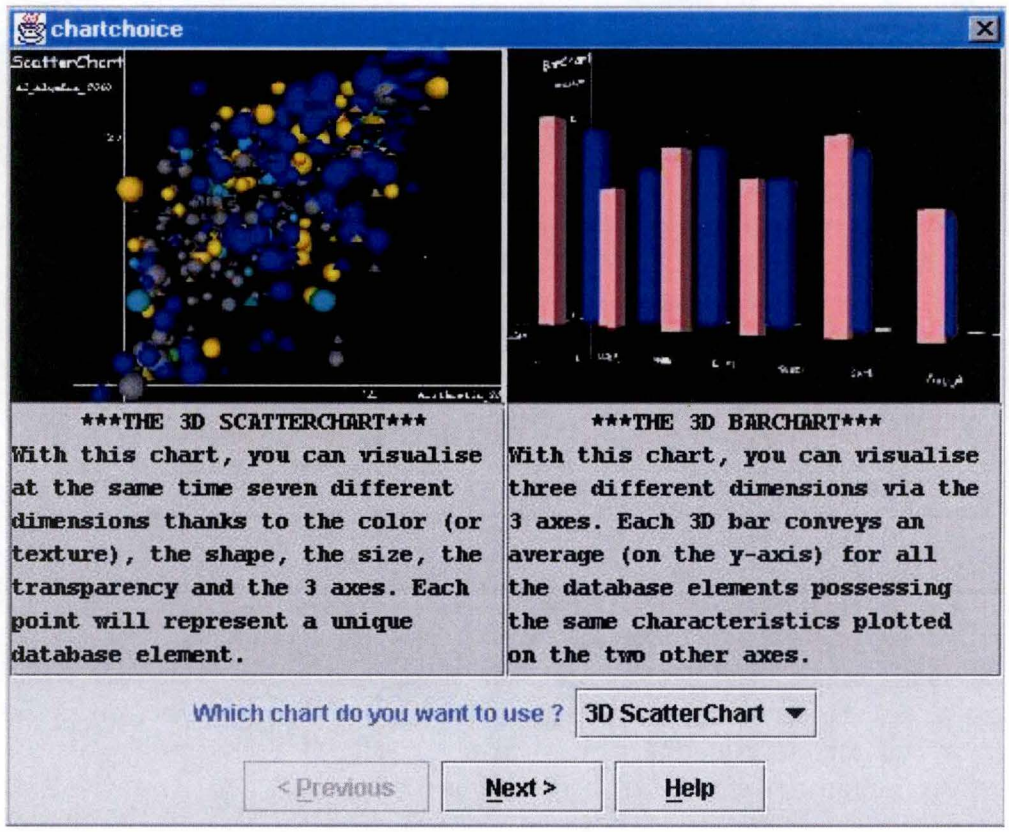

Figure 34a: The first frame of the interface

Regardless of the specific chart selected, the others frames will not really change. For the *ScatterChart*, they will have the following appearance:

The second frame (Figure 34b) allows the user to choose the columns that he wants to visualize. It is also here that he can enter a title for the chart. The third and the fourth frames will depend on this one.

In the third frame (Figure 34c), the user selects his constraints or filters. That allows him to choose which students he wants to represent on his *ScatterChart*. If he does not type any constraint, more than 1000 points will appear. This will affect the legibility of the chart. For example, if you want to see only the results for male students, you must select in front of the label "**Gender**", the choice that allows such a selection. You could also just want to visualize the students that have an average over fifty percent and so on.

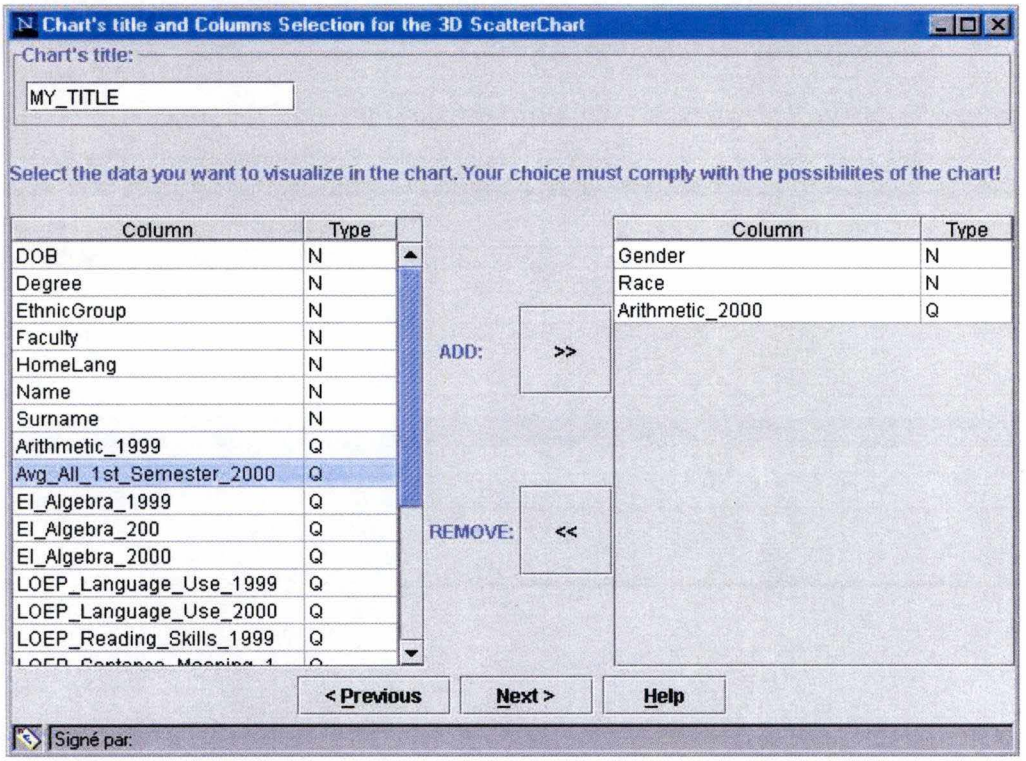

Figure 34b: The second frame of the interface

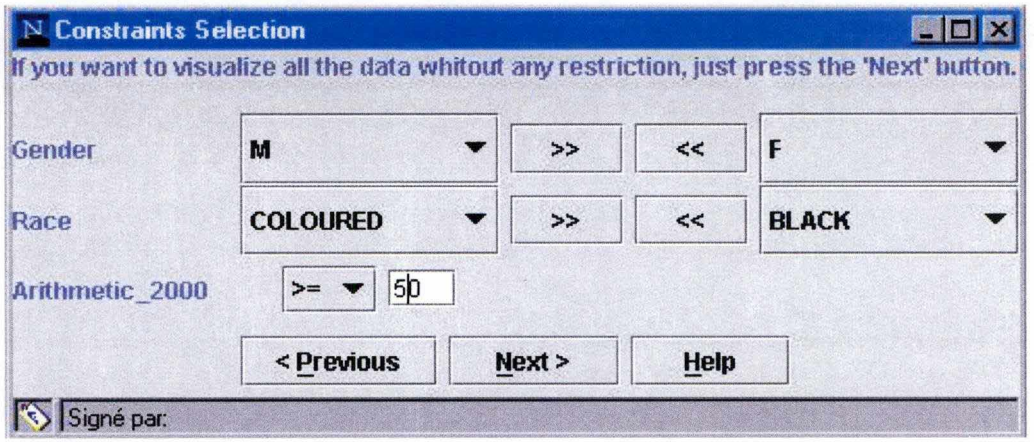

Figure 34c: The third frame of the interface

In the fourth frame (Figure 34d), he chooses which dimension he will use to represent each column. He only has the possibility of associating dimensions applicable to the data type of the column. If he chooses the Shape, the Colour, the Transparency or the Texture dimension, then another frame (Figure 34e) is opened. It enables him to associate each distinct value of this column with a representation. For example, if he has associated the Gender with the colour, he will have to associate "M" and "F" with two different colours.

Eventually, to visualize the chart, he simply has to press Finish in the fourth frame after having completed to associate each column. Once the virtual chart is displayed, he will no more be able to modify it via this interface. An interesting improvement of the *Visualization Application* could provide the user with another interface. This new interface will be embedded in the virtual world, and will allow the user to modify in real time the main parameters of a chart. For instance, if the chart contains too much information, he will be able to choose only a subset of the current chart. Or, if he is not happy with a dimension, he will be able to choose another one more representative.

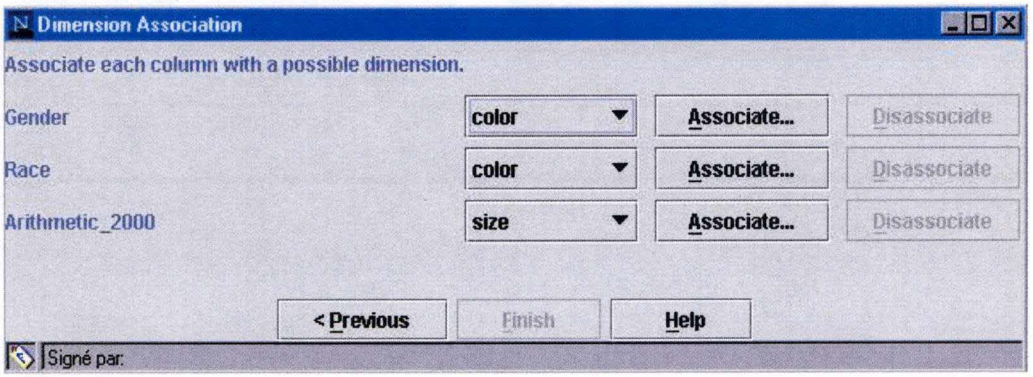

Figure 34d: The fourth frame of the interface

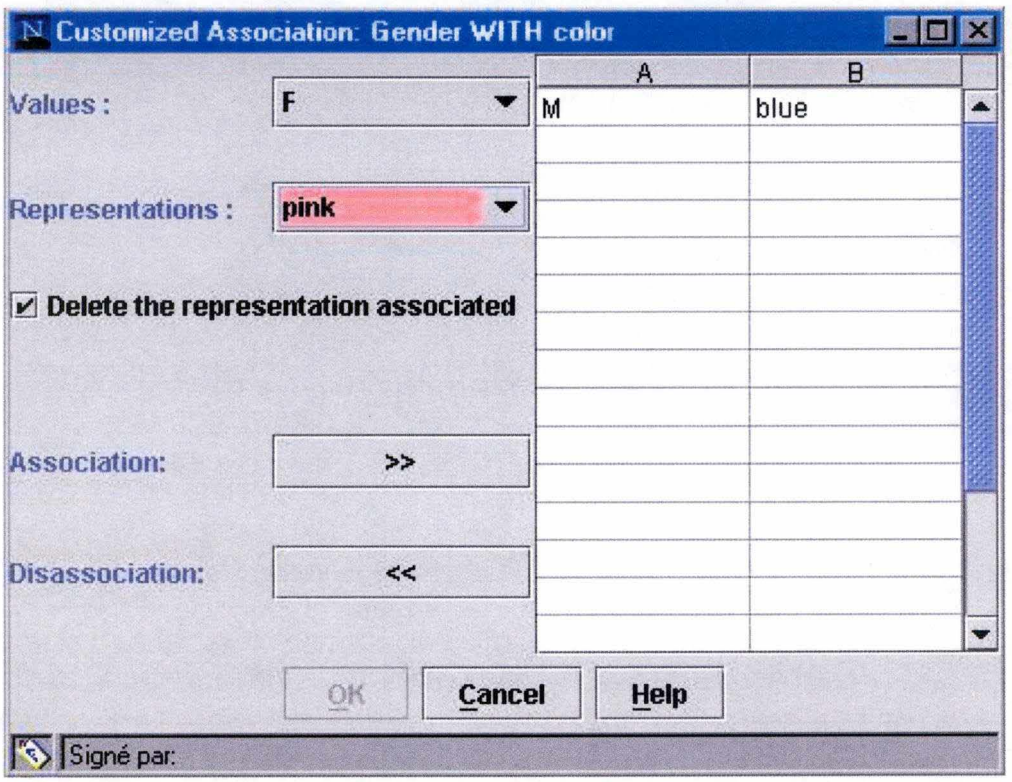

Figure 34e: The fifth frame of the interface

VI. Design of the interface

After having presented the general aspect of the User Interface in the previous section, we explain and justify the choices we have made. Some of these choices were imposed by the nature of the *Visualization Application* and others were made deliberately. Some of them can raise some problems of efficiency for the user. Anyway we have tried to help a maximum the user in his task. Nevertheless, several things have not been implemented by lack of time. We have also tried to comply a maximum with the design criteria. For the sake of completeness, an evaluation of this interface is proposed in the appendix A.

VI.1 Imposed Choices

To display the customised chart that the user has chosen, he must achieve several sub-tasks. He accomplishes each of them by means of a specific frame (cf. the previous section). The only exception is the second frame where the user has to give the chart a title **and** to select the data he wants to visualize. The user can concentrate all his attention on a task at the same time, and is not faced with too much information.

If we use five frames to materialize the global task instead of one tab pane that would contain all these frames, it is due to the logic of the application. Indeed, each of these sub-tasks must be accomplished in a particular order because one particular frame exerts influence on the next one. So, in the second frame (see Figure 34b) the overall number of columns that the user can choose will depend on the choice he has made in the previous frame (see Figure 34a) : he has chosen to visualize either the *3D ScatterChart* or the *3D BarChart*. Next, the second frame allows him to choose the columns that he wants to visualize. The third (Figure 34c) and the fourth frames (Figure 34d) will depend on this one. In these two frames, the user will be able to perform an action for each selected column previously. Finally he may get to the fifth frame (Figure 34e) via the fourth one if the association needs to be customized. This last frame will enable the user to associate each distinct value of the associated column with a VRML representation. Actually, he has no need to associate each distinct value, but only the ones he has selected in the third frame.

The possibility to manipulate different databases was the other constraint with which we had to cope. That is why we use representation structures (like the **JComboBox**[28], or the **JTable**[29]) that can display as much data as possible, rather than other structures (like the **JCheckBox**[30]) that are more suitable in the case where there are only a few different possibilities. For example in the third frame (Figure 34c), we use two **JComboBox** structures and two **JButtons**[31] rather than

[28] A JComboBox is a drop-down list of items from which the user can make a selection by clicking an item in the list or by typing into the box, if permitted.

[29] A JTable is a table that can contain several different columns, with an unlimited number of data.

[30] A JCheckBox is a Graphical User Interface component that is either selected or not selected.

[31] A JButton is an area that triggers an event when clicked.

two **JCheckBox** structures (Male and Female) to enable the user to choose the gender(s) he wants to visualize.

Since the application runs with any database, we cannot have a priori knowledge of the values that the database contains. This is another problem. For instance, if the column's names or the different values of these ones are not relevant, the user will have some difficulties to understand their meaning. The *Visualization Application* does not alter the data but simply enables the user to select and visualize some of them.

VI.2 The interface must facilitate the user's task

As already said above, the user accomplishes each of the sub-tasks by means of a different frame. To make easier these sub-tasks, the user must not have the possibility to make execution errors, and as little as possible intention errors. To prevent these errors, the user cannot perform an action that soon or later, would involve an illegal operation. So for instance, in the second frame (Figure 34b), the user's choice must comply with the possibilities of the chart. If it is not the case, a message (Figure 35) inform him, and he cannot proceed to the next frame. If he wants to go on, he must alter his selection. So the errors are treated and the user is immediately and accurately notified.

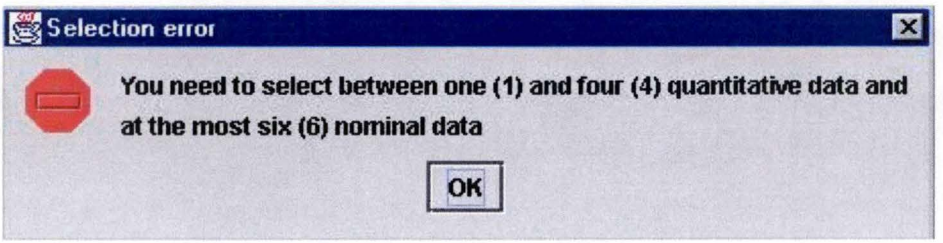

Figure 35: Message "Selection error"

To prevent these errors, we also use another way. We do not enable the user to do what he is not authorised to do. That is to say we disable the **JButtons** and other components that he cannot use. This technique guides also the user in his task and makes the task easier. For example, the *Finish* **JButton** (Figure 34d) is disabled as long as all the columns have not been associated with a dimension. Also, only the columns that have not yet been associated, have their *Associate...* **JButton** enabled. Since he can fill in a particular frame in the order he wants, this kind of feedback was necessary so that the user knows about the progression of the task. It is particularly important in these cases for the user to know what he has already done and what he has still to do. Furthermore, the task can also have been interrupted. Finally, when he detects an error, he can easily correct it inside each frame.

In order to correct his errors or simply if the user changes his mind, he can return to the previous frame. He can so easily go from one frame to another one via the *Next* and *Previous* **JButtons**.

Another important point considering the length of the task is the presence of *inter-referential inputs and outputs*. The inputs of a frame appear as outputs in the next frame(s). It allows the user to interrupt the task and to help him in the task's achievement.

VI.3 Deliberate choices

We present here the different choices we have made concerning the design of the interface. Of course, there exist so many different ways of building an interface. That's why we attempt below to justify these choices.

We could have chosen to make one frame with the third and the fourth ones. In this case, the user would have in front of him more information simultaneously. It would enable the user to correct easier his choices : he would not need to return from the fourth frame to the previous one to change the selection of the constraints. But by merging the frame, we would have grouped two different problems : the selection of the constraints and the association of the columns. We have preferred not to mix the user's sub-tasks. It seems to us that *"one main sub-task, one frame"* is a better idea for the design.

In the fifth frame (Figure 34e), we have decided to add a **JCheckBox** that enables the user to choose if he wants or not to delete the representation, once this one has been associated with a value. In fact, if the user associates (in the fourth frame) a column with a dimension that contains less distinct representations than the number of distinct values in these column, we notify immediately the user about it. Then, he can customize the association in the last frame. At this time, he will have to associate different values with the same representation. So, concerning the association between a column and a dimension, he is not limited by the number of distinct representations this dimension contains. By default, if there are enough different representations, he cannot associate two values with the same representation : it is not advised. However he can do it if he wants (by changing the default value of the **JCheckBox**). So the application guides the user without being inflexible. The user's satisfaction can only be higher.

Likewise, in the fourth frame for the *ScatterChart*, the user has the possibility to associate a column having a nominal type with one of the three axes. Although this choice is not the best, he could opt for it if he wants to compare more easily the different elements of a column. He could also do this choice if he selects more than three nominal columns, because only three of them can be associated with the colour (or texture), the shape and the transparency. The application enables again the user to do what he wants but it can raise some problems. So, the user could display a confused *ScatterChart* in which most of the data would be superposed. Therefore, he can sometimes have a bad surprise. But, it was difficult for us to do differently if we want to enable the user to choose his database and to offer him a flexible application.

As we have said, the application try to make easier the user's task. It is in this aim that we show him interactively each shape or texture he wants to choose (in the fifth frame). The chosen colour appears also immediately in the **JComboBox** (cf.

Figure 34e, where you can spot the pink colour). This can aid him to better remember the associations and lighten the interpretation of the chart. We are aware that there exist other manners to do better but we have not applied them by lack of time. It is also by lack of time that we have not implemented the same thing for the transparency dimension.

In fact, the major problem is that the user do not see the chart before he has completed the whole task. Unfortunately, it is only at this time he will be able to detect his intention errors. The aim of the graphics above the chart's selection in the first frame (Figure 34a) is just to give him a general idea of the two possible charts. It will be mostly useful for the first utilizations of the *Visualization Application*.

Finally, mind you that the user cannot yet propose a new database via the interface. The current database's name is **DataStudents** (this is an Access database, accessible via the Microsoft Access Driver pilot[32]) and the table's name is **Results**. So the user must denominate the new database **DataStudents**, the new table **Results**, and must put the database in the same directory as the previous one. Of course, the first thing to add in the future to the *Visualization Application*, is to allow the user to propose a new database via the interface, just by giving the name and the location of this database.

VI.4 Problems

When the user wants to correct some choices he previously made in another frame, he has just to press the *Previous* button. In fact, pressing this button has the same effect as cancelling an action. So he loses the information he has just entered. It is particularly awkward when he returns from the fourth frame to the third. The problem would be resolved for these two frames if they were merged, but we have kept the first solution for the reasons evoked in the previous paragraph. If we had more time, we would have been able to prevent or at least to limit this loss of data.

VI.5 Link between the interface design and the *Functional Architecture*[33]

The link is simple. What we call a window in the *Functional Architecture* (FA) is represented by a frame in our interface. So the window W_{11} (cf. Figure 44, page 118) in the FA is represented by the first frame, W_{12} by the second one, W_{13} by the third one, W_{14} by the fourth one, and W_{15} by the fifth one. This correspondence is logical since we have decided to use a different frame for each one of the sub-tasks that the user must perform.

---

[32] To connect to the database, an *ODBC data source* must be registered with the system through the **ODBC Data Sources** option in the Windows **Control Panel**. *ODBC* is a technology developed by Microsoft to allow generic access to disparate database systems on the Windows platform (and some Unix platforms). The Java 2 Software Development Kit comes with a driver to allow any Java program to access any *ODBC* data source.

[33] The *Functional Architecture* is presented in the chapter six.

VI.6 Used design criteria (according to [Bodarta1998])

- **The compatibility** : the compatibility is interpreted like a coherence with the application's outside environment (the user's expectations, his behavioural habits and others). The aim is to reduce the need to translate, to interpret the information into system data, and to shorten the task's interpretation into action. The task's completion will be in this case faster. This interface pays attention to the behavioural and operational compatibility : it is compatible with the user's expectations and the order of the operations. The order of the operations has been discussed in the paragraph VI.1.

- **The mental strain** : it is a specially important criterion in the *Visualization Application*. We take it into account : the quantity of data to handle and actions to perform by unit of task is reduced. The user is more efficient during his sub-task's realisation when he is less distracted by information concerning others sub-tasks. With the current interface, as we have pointed it out before, the user can concentrate on one sub-task at the same moment.

- **The adaptability** : the *Visualization Application* is not adaptable. The user has no alternative ways to perform his task.

- **The dialog control** : the interface endeavours to provide the user with the feeling that it is placed under his control by executing the actions in response to explicit user's requests. The aim is to let him control the sequence of dialog as much as possible.

- **The guiding** : this point is already mentioned in the paragraph VI.2 where we talk about the used ways to make easy the user's task. The interface informs the user of his action's result and of his position in his task's completion. The feedback is indeed immediate and easy to interpret. The aim is to give the user a help about what he can do, about the situation in which he finds himself, and about the results of the performed actions. We reach to provide this help by indicating the user the buttons and other window components he can or cannot use.

- **The errors management** : we have talked about the execution and intention errors in the paragraph VI.2. To avoid the errors as much as possible, we use the same trick as for the guiding : we indicate the user the actions he can or cannot do. If his choice is not correct, he won't go on this sub-task and will have to begin it again. Furthermore, the interface withstands errors made by the user and enables him to correct them easily. A last remark, also valid for the guiding : the user can go from one frame to another one without any difficulty by using the standard *Previous* and *Next* buttons.

VII. Conclusion

To conclude, we are going to present a general architecture for the *Visualization Application*, that includes all the main components necessary for the good course of the application, that is the *User Interface*, the *Charts API* (Applications Programming Interface), the *Database* component, and the *VRML Browser*.

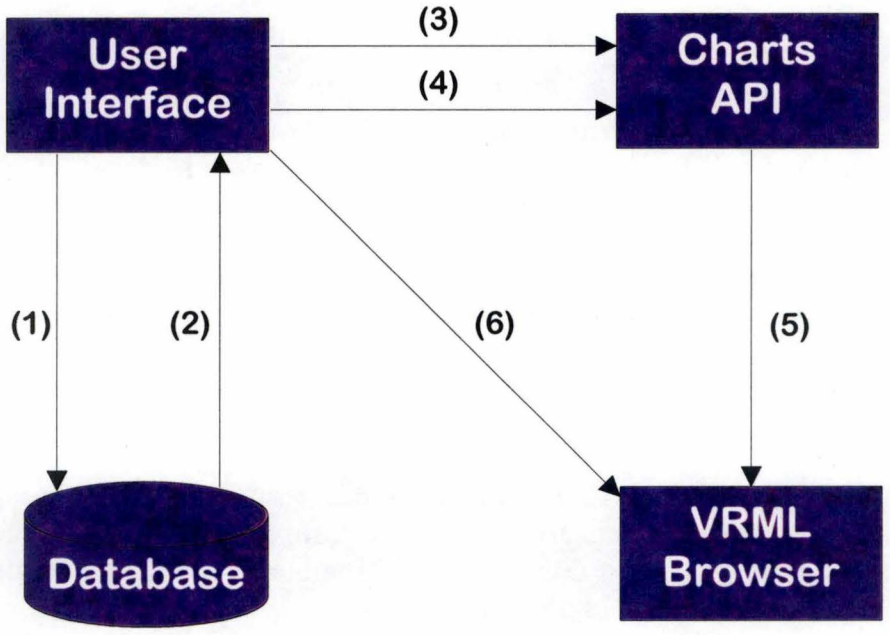

Figure 36: General Architecture

The *User Interface* includes all the components we have presented in the *Prototype of the Interface* section. The entry point of this interface is a Java Applet, needed to initialise the *VRML Browser* (embedded in the Web Browser, namely Netscape Communicator, in which the Applet will be loaded) with a virtual world containing only three axes (x, y and z). Then, the task's remainder that the interface proposes is undertaken by the chaining of the different frames (cf. section V). This interface has also to know about the semantics of the different charts it proposes to visualize.

The *Database* component concerns the data set the user wants to visualize, and allows the *Visualization Application* (via the interface) to access and to submit queries to this data set.

The *Charts API* refers to the different classes that are directly connected with the charts' generation in a virtual world. That is to say, in the case of the current application, the *Chart*, the *ScatterChart*, the *BarChart*, the *Datum*, the *Representation*, and the *Semantic* classes.

Eventually, the *VRML Browser* regards the *Virtual Environment* in which the chart will be visualized.

Now we can comment upon the previous architecture, via the numbered arrows visible in Figure 36.

(1) The *User Interface* access the database, by submitting SQL (Structured Query Language) queries. The user, without knowing it, creates the main query with regard to the data he wants to visualize, through the interface. The other queries are automatically created by the interface, for example to know about the data types, or to know about the different values of a variable, and so on.

(2) This point only concerns the queries' results returned to the interface.

(3) Before the building of the virtual chart, the interface needs to initialise the looked-for chart, in order to create the legends and to prepare the dynamic chart's generation. This is achieved by passing as parameters to the specific chart class (*ScatterChart*, or *BarChart*), a vector of *Semantic* objects and the chart's title (cf. paragraph II.1).

(4) When the initialisation is completed, the interface can send to the *Charts API* a row of data at the same time. These data arise from of the main query created implicitly by the user (cf. point 1 just above). Each row (or group of rows) of data will be transformed as a VRML object, and will be ready to be visualized in the virtual world.

(5) Here, the interface, via the *Charts API*, will build dynamically the virtual chart into the *VRML Browser*.

(6) Finally, this arrows materializes the initialisation of the *VRML Browser* with a virtual world containing only three axes (x, y and z).

For the possible improvement of the *Visualization Application* we have presented in the fifth section (cf. page 74), it will directly concern the *VRML Browser* component. Indeed, the new interface will be embedded in the virtual world, and the user will be confronted at the same time to his virtual chart and to this interface. Any modification will be immediately perceptible, and the application will be more flexible. So, the analysis may be faster and more effective.

The implementation of this new interface is quite long. In fact, it needs to remember of all what has been done during the customisation of the chart (via the *User Interface*) in order to be able to erase entirely the previous chart and to display the new one. Because, if you can add objects in a virtual world as often as you want, you cannot erase or modify only certain objects of this world. That is why, by lack of time, we did not manage to offer this interesting possibility to the user.

Chapter Four

Implementation

I. Introduction

The purpose of this chapter is to discuss the choice of the implementation tools and discuss the implementation of the application.

As our project is a validation of Wesson's arguments that *Java* technology is mature enough to provide the tools for data visualization, the main implementation tool will be *Java*. As discussed briefly in the Introduction chapter, we will also make use of the *VRML*[34] technology.

Let's also keep in mind that our task is a continuation of Darville and Van Espen's study, and that they have also essentially used these tools.

II. Implementation tools

The technology chosen for the implementation platform is the same as the one chosen by Darville and Van Espen. In their research, they grouped the technological requirements for their application in eight points.

1. It is essential for the technology to allow the dynamic building of charts as well as the programming of some behaviours.
2. It is also essential to support the display of three-dimensional contents.
3. There is a need for navigation and interactivity.
4. The availability of basic elements found in today's 3D applications.
5. The ease of development (simplicity and reusability).
6. Usable and useful Graphical User Interface.
7. Performance.
8. The universality (file interchange format, cross-platform and network-centric applications).

They have already shown in [Darville&2000] how the two technologies comply with these requirements. But for the *VRML* technology, he have deepened the researches concerning the requirements for the *Virtual Reality*.

[34] This technology will be introduced later in this chapter (cf. II.3).

We can also add that as we are going to manipulate large data sets from databases, the technology has to allow us to interact with such databases in an easy way. The choice of *Java* and *VRML* will allow the development of sophisticated and robust applications as well as dynamic, interactive, three-dimensional contents.

II.1 *Virtual Reality* Requirements

Dr. Kulwinder Kaur is a researcher at the *Centre for Human-Computer Interface Design*, City University, at London. She has investigated usability guidance for designing and evaluating virtual environments. Once the investigation completed, she has translated the Virtual Environments' design requirements into concrete design guidelines. Now, we are going to present these guidelines [Kaur1999].

a) Guidelines concerning the design of objects

Objects are elements in the Virtual Environment which are seen by the user to individually possess functionality and meaning. For the design of objects, how each object will be represented in the VE needs to be detailed and the following guidelines apply.

- **Make objects easy to distinguish**

  Design guideline: Objects should be easy to distinguish from all likely viewpoints. Object representations should be distinctive with clear boundaries. Important parts of the object should be easily distinguished from the object image.

  Motivation: Necessary to enable the user to perceive individual objects in the environment.

  Context of use: More consideration should be given in cluttered environments or where there are many similar looking objects. More consideration should also be given to the more important objects in an environment.

- **Make objects easy to identify**

  Design guideline: Objects should be easy to identify or recognise. Individual parts of an object, particularly interactive parts, should also be easy to identify and prominent features of objects should be represented. Objects modelled on real world phenomena should be represented accurately and appropriately to match any expectations the user has.

  Motivation: Necessary for the user to know what the different objects in the environment are for him to understand and interact effectively with the environment.

  Context of use: More consideration should be given to abstract objects (that are not modelled on real world phenomena) or in cases where the user may not have much prior knowledge about an object's identity. More

consideration should also be given to the more important objects in an environment.

- **Make the interactivity and significance of objects clear**

  Design guideline: It should be clear to the user whether or not objects can be interacted with. It should also be clear whether objects have the ability to act independently of the user or initiate interactions with the user. The relative importance of objects in the environment and to the user task should be made clear.

  Motivation: Provides important information to aid the user understand objects during exploration and plan interactions on interactive objects.

  Context of use: More consideration should be given to abstract objects (that are not modelled on real world phenomena) or in cases where the user may not have much prior knowledge about an object. More consideration should also be given to the more important objects in an environment.

- **Make objects easy to access**

  Design guideline: Objects should be easy to access, that is, it should be easy for the user to approach objects and take up a suitable position close to objects.

  Motivation: Necessary when the user is approaching objects and orienting to objects for investigation or for carrying out actions.

  Context of use: More consideration should be given in cluttered environments, where the user is navigating in very restricted areas, or where the user has limited navigation pathways, such as when objects cannot be passed through. More consideration should also be given to the more important objects, such as those that can be interacted with, to the smaller objects and to objects whose position or orientation in the environment can change.

b) Guidelines concerning the design of user actions

User actions are sets of operations or activities that the user can carry out with objects, thereby affecting changes in the environment. For the design of user actions, how each action will be represented, how the user will carry out the action and how the effect of the completed action will be represented needs to be detailed and the following guidelines apply.

- **Show what actions are available**

  Design guideline: The availability for action should be made clear to the user.

Motivation: Necessary to aid the user in finding available actions during exploration.

Context of use: Particularly important for exploratory applications. Less applicable where the user has information about actions available in the VE, for example as in the case of VE 's accurately modelling activities in a domain well known to the user.

- **Make the purpose of actions clear**

  Design guideline: It should be made clear to the user what each action is for, that is the resulting effect of an action, and this should meet any expectations the user may have.

  Motivation: Aids the user understand the purpose of each action. This is necessary during exploration and when the user is assessing the relevance of actions to the his task and goals.

  Context of use: Particularly important for exploratory applications. Less applicable where the user has information about actions available in the VE, for example as in the case of VE 's accurately modelling activities in a domain well known to the user.

- **Show how to carry out actions**

  Design guideline: The sequence of operations required to carry out actions should be clearly defined and match any expectations the user may have.

  Motivation: Essential for the user to know what operations are required to carry out actions. Avoids forcing the user to resort to guesswork when carrying out actions.

  Context of use: Particularly important with more complex action sequences, involving multiple operations or components, or a choice of interaction styles such as click/drag/double-click. Less applicable where the user has information about the action sequence, for example as in the case of VE 's accurately modelling activities in a domain well known to the user.

- **Make actions easy to execute**

  Design guideline: Actions should be efficient to execute and there should not be frequent obstacles/problems in executing actions. The action sequence should be as simple as possible. The demand of manipulation precision and motor co-ordination should be within usual human ability.

  Motivation: Essential for the user to be able to execute actions. Also, the user needs a high level of control over the action execution so that he can easily achieve the effect he wants.

Context of use: Generally necessary for all actions. Particular consideration needs to be given for actions sequences that have a long duration or complex actions which may require precise and difficult manipulations. More consideration also needs to be given for more common actions, such as navigation.

- **Show the effect of completed actions**

  Design guideline: Feedback on the effects of completed actions should be given and should be easy for the user to distinguish. The feedback should be timely, accurate and integrated across all modalities (vision, sound etc.) used.

  Motivation: Important that the user can see the effects of actions that he has carried out, so that the success of actions can be assessed and any errors detected.

  Context of use: Generally required for all actions. More consideration needs to be given for complex actions with multiple or varying effects.

c) Guidelines concerning the design of system control

System control involves the system or an environment agent taking control over some part of the user's interaction. Later, control is generally returned to the user. For the design of system control, how the control will be represented to the user needs to be detailed and the following guidelines apply.

- **Show that control has begun or ended**

  Design guideline: Whenever the system or an agent takes control of interaction from the user, this should be made clear. It should be clear when control has been returned to the user.

  Motivation: Essential for the user to be aware when he no longer has control over some part of his interaction.

  Context of use: Generally applicable. More important where significant parts of the user's interaction are affected, such as navigation.

- **Show why control has taken place**

  Design guideline: The goal of the system in controlling the interaction should be made clear to the user. There should be a clear indication to the user when control is likely to be returned to him.

  Motivation: Necessary for the user to understand why the control is taking place so that he can benefit from it in any intended ways. Important for the user to be aware how long the control will last so he can plan future interactions and plan whether he should attempt to regain control.

Context of use: More important where significant parts of the user's interaction are affected, such as navigation.

- **Show what actions are available during control**

  Design guideline: Any actions available to the user during system control should be made clear.

  Motivation: Necessary to enable the user to plan interactions during system control and especially plan for regaining control.

  Context of use: More important where significant parts of the user's interaction are affected, such as navigation, and with longer duration of control. More important for actions affecting control, such as actions to terminate the control.

All these guidelines are applicable to Virtual Environments and to their design. But for the purposes of the *Visualization Application*, the tool that will allow us to visualize the charts in virtual worlds does not need to comply with all these requirements, even if it complies.

Now, we will give the guidelines that are essential for the application, and see that the Cosmo Player[35], a *VRML* browser, fulfils these guidelines.

First, Cosmo Player *makes objects easy to distinguish*. In the different screenshots (Figure 39, 41, 42 and 43) we can distinguish easily the different objects, with regard to their colour, their position, and their shape. Of course, the user has to give the objects values that allow such a distinction. If he chooses to represent some objects with a given transparency, and others with a transparency not so different from the first one, he would not be able to distinguish these objects.

Then, the same characteristics presented above allows to *identify objects without problems*. In this case also, the user has to give the objects pertinent values. If he chooses to represent some objects with a maximum transparency, or with the black colour, he would not be able to see these objects.

Next, the navigation[36] tool (cf. Figure 37) that will be discussed below, makes *objects easy to access*. It is quite intuitive and easy to use. Besides, the predefined viewpoints[37] allow the user not to get lost in the virtual world, and facilitate the navigation within the world.

---

[35] Cosmo Player is the *VRML* browser we will use for the application (cf. II.3).

[36] Navigation : movement and directing of movement through a space, such as a *virtual environment.*

[37] Viewpoint : position in the *virtual environment* from which the user receives environment output, such as the image and sounds.

This navigation tool *show exactly and at any time what actions are available.* Of course these actions concern only the navigation throughout the world. The other possible action is to ask for exact values of an object, and the legend tells the user how to accomplish this: he only needs to drag the mouse on the looked-for object.

After that, the same tool *makes the purpose of actions clear* and *shows how to carry out actions*. In fact, for each possible action, it gives the user its purpose: he only needs to drag the mouse on the button that materializes this action. Furthermore, it provides the user with help facilities, easily accessible.

To *execute the actions* the user does not need a manipulation precision and a motor co-ordination above usual human ability. He just has to manipulate the mouse as he is used to.

II.2 The Java Technology

For the dynamic chart generation and behaviour programming, an API will be written in *Java* in combination with the External Authoring Interface (EAI).

- The *Java* programming language is a strongly typed, object-oriented language that borrows heavily from the syntax of C++ while avoiding many of the C++ language features that lead to programming errors, obfuscated code, or procedural programming [Singhal&98].

- The EAI is a scripting language integrated within *VRML*. It allows the programmer to control from a *Java* applet running in a web page (display by a Web browser) the content of a *VRML* browser window embedded on the same web page. The EAI is a proposed Informative Annex to the *VRML* specification [Darville&2000].

The most popular style of database system on the kinds of computers that use *Java* is the *relational database.* Object-oriented databases have also become popular in recent years. A language called *Structured Query Language* (SQL) is almost universally used among relational database systems to make queries (i.e., to request information that satisfies given criteria). *Java* enables programmers to write code that uses SQL queries to access the information in relational database systems. Some popular relational database software packages include Microsoft Access, Sybase, Oracle, Informix and Microsoft SQL Server [Deitel&1999]. The *Java* Database Connectivity (JDBC) model will be used to manipulate a Microsoft Access Database.

The AWT[38] and Swing components of *Java* (JDK 1.2) look very suitable for the Graphical User Interface (GUI) development. In fact, the *Java* run time includes class libraries that provide high-level interfaces for GUI programming.

---

[38] AWT corresponds to the original components from the *Abstract Windowing Toolkit* package. In contrast with *Swing* components (also referred to as *lightweight components*), the AWT components (called *heavyweight components*) are tied to the local platform. In other words, they rely on the local platform's windowing system to determine their functionality and their look and feel. [Deitel&1999]

As one of our main objectives is to achieve a certain reusability, the tools we use have also to provide the application with an adequate portability. *Java*, as one of these tools, complies very well with this objective. So *Sandeep Singhal* and *Binh Nguyen* declare in [Singhal&98]:

> "The 'WRITE ONCE, RUN ANYWHERE' slogan is synonymous with the Java Programming language. The Java run-time library provides application developers with the ability to write code in one single language and the confidence that the code will execute without modification on virtually any hardware, any operating system, and any application environment".

Of course, *Java* is not the only system that provides such a platform independence, but this is the first to achieve widespread popularity among commercial developers. According to [Tyma1998],

> "Platform independence in Java really takes two forms. The popular notion is that we write our code, compile it and then never worry about having to port it to new machines (write once, run everywhere). The more overlooked side of platform independence is Java's rather fantastic abstraction of many programming paradigms".

II.3 The *VRML* Technology

VRML[39] is an acronym for the Virtual Reality Modelling Language. This language is to 3D what HTML is to 2D. It is a web-centric technology, which allows the user to describe interactive three-dimensional worlds. To be able to visualize those worlds we need a *VRML* browser that interprets and renders them in real time. Usually, the browser provides a navigation tool to move within the world.

VRML defines most of the commonly used elements found in today's 3D applications. It is an effective 3D file interchange format that can be used to describe a wide variety of 3D scenes and objects. It is not a programming language like *Java*, but a descriptive language. The *VRML* specification was recognized as an international standard by the International Organization for Standardization (ISO) and the International Electro-technical Commission (IEC) in December, 1997 [VRML97].

Long before its official standardization *VRML* became the de facto standard for sharing and publishing data between CAD, animation, and 3D modelling programs; virtually every one of those programs now exports *VRML* or has a utility or plug-in to convert its native file format to *VRML*. *VRML* is included or referenced in the upcoming MPEG-4 standard, Java3D, and in other developing standards.

---

[39] VRML is a file format for describing interactive 3D objects and worlds. VRML is designed to be used on the Internet, intranets, and local client systems. VRML is also intended to be a universal interchange format for integrated 3D graphics and multimedia. VRML may be used in a variety of application areas such as engineering and scientific visualization, multimedia presentations, entertainment and educational titles, web pages, and shared virtual worlds. [Hang&1998]

VRML has been designed to fulfil the following requirements [Hang&1998]:

- **Authorability** :
  Enable the development of computer programs capable of creating, editing, and maintaining *VRML* files, as well as automatic translation programs for converting other commonly used 3D file formats into *VRML* files.

- **Composability** :
  Provide the ability to use and combine dynamic 3D objects within a *VRML* world and thus allow reusability.

- **Extensibility** :
  Provide the ability to add new object types not explicitly defined in *VRML*.

- **Implementability** :
  Capable of implementation on a wide range of systems.

- **Performance** :
  Emphasize scalable, interactive performance on a wide variety of computing platforms.

- **Scalability** :
  Enable arbitrarily large dynamic 3D worlds.

VRML is capable of representing static and animated dynamic 3D and multimedia objects with hyperlinks to other media such as text, sounds, movies, and images. *VRML* browsers, as well as authoring tools for the creation of *VRML* files, are widely available for many different platforms.

The aspect of motion in *VRML* is really important as we are going to see it. 3D does not lend itself to rigorous comparative analysis because of the distortion arising from a perspective view and occlusion. However, by using *motion and animated interaction*, it is possible to use 3D as a reliable, accurate and precise decision-support tool.

> "To make 3D work, you need to make it move". This is what Wright calls '**Information Animation**' in [Wright1999].

VRML allows the developer to give a world a huge set of visual effects (colour, texture...) and behaviours (like animation). Exploring a *VRML* world is very easy because many free browsers are available on the Net. The browsers are often powerful and offer interesting capabilities. *VRML* 3D browsers are typically installed as plug-ins within 2D browsers (such as Netscape Navigator and Internet Explorer).

VRML Objects exist in a virtual world and can be linked to various objects such as *Java* applets, images, multimedia files and even URL's. Together with *Java* and due to its capabilities and its simple architecture, *VRML* is a suitable development platform for our Information Visualization purposes.

The *VRML* browser we will use is the *Cosmo Player* from *Cosmo Software*©. The screenshot displayed on Figure 37 presents the navigation tool (or the dashboard) of this software. This tool is really easy to learn to interact with. The first picture shows the controls to move around in the world. The second one shows the controls to examine objects in the world. The third one shows the change controls to switch between the two previous ones. And the last one shows how to move to predefined viewpoints into the world.

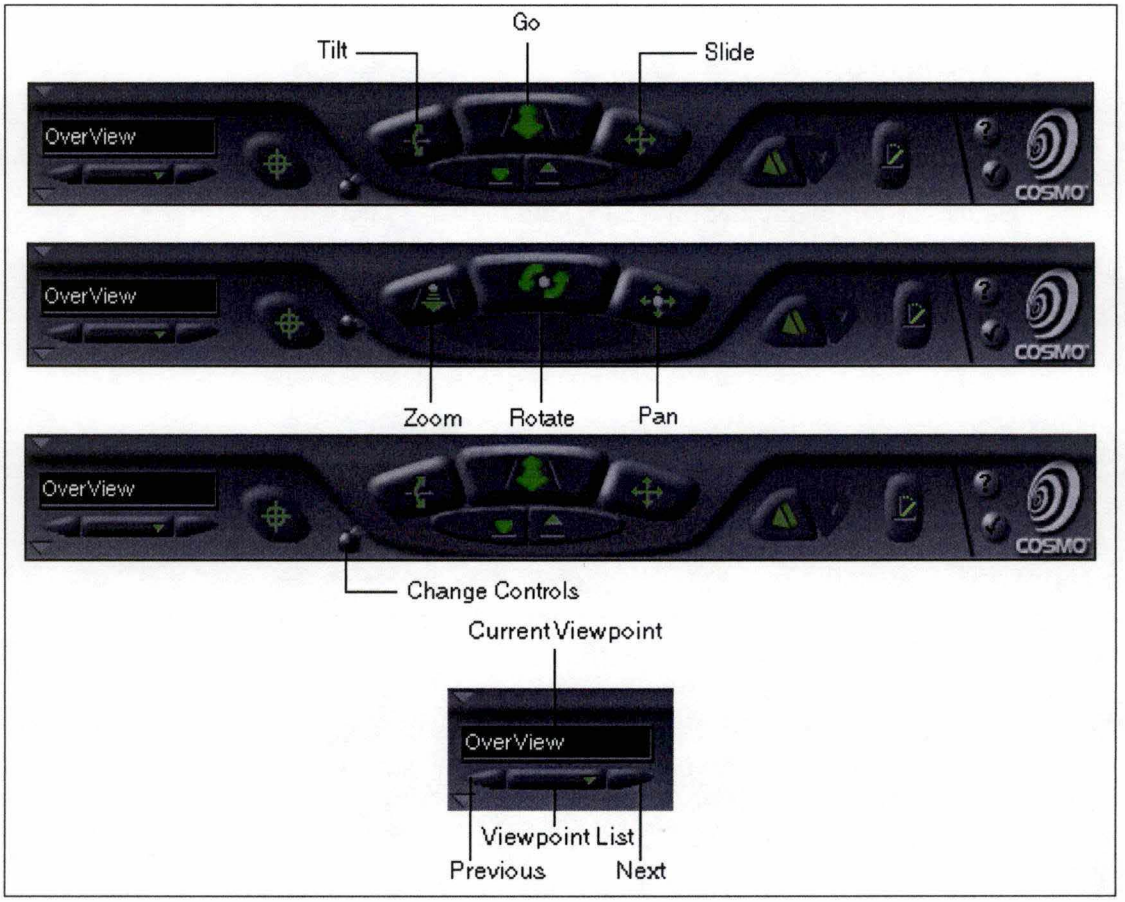

Figure 37: The navigation tool of *Cosmo Player*

II.4 Why *JAVA* and *VRML* together?

Over 20 million *VRML* browsers have been shipped with Web browsers, making interactive 3D graphics suddenly available for any desktop. *Java* adds complete programming capabilities making *VRML* fully functional and portable. This is a powerful combination, especially as ongoing research shows that *VRML* plus *Java* provide extensive support for building large-scale virtual environments. The two next extracts show precisely why we have chosen to work with these both technologies.

"Integrated with the Virtual Reality Modelling Language for describing interactive 3D scenes, Java allows the introduction of dynamic behaviour models for VRML objects and multi-user interactions in these virtual worlds". [Singhal&98]

"Integrating two powerful and portable software languages provides interactive 3D graphics plus complete programming capabilities... Using Java is the most powerful way for 3D scene authors to explore the many possibilities provided by VRML... They (Java and VRML) are well matched, well specified, openly available and portable to most platforms on the Internet. VRML scenes in combination with Java can serve as the building blocks of cyberspace... Using Java and VRML, practical experience and continued success will move the field of virtual reality past speculative fiction and isolated islands of research and onto desktops everywhere, creating the next-generation Web". [Brutzman98]

The use of *Java* and *VRML* will allow us to build an application that produces interactive 3D graphics, via complete programming capabilities. These capabilities are offered by *Java* and the External Authoring Interface (EAI), by means of the class libraries they supply. Besides, thanks to some of these libraries (the *AWT* and *Swing* Components), we will be able to propose to the user a nice and suitable interface.

As mentioned earlier, the EAI defines a *Java* or *JavaScript* interface for external applets that communicate from an "external" HTML Web browser. EAI applets can pass messages to and from *VRML* scenes embedded in an HTML page. The primary benefit of the EAI is the ability for direct communication between the encapsulating HTML browser and the embedded *VRML* browser. For example, the communication between Netscape Navigator (the HTML browser in the *Visualization Application*) and Cosmo Player (the embedded *VRML* browser we use) constitutes the core component of our application. In fact, without this, we would not be able to interact with the virtual world and display the 3D charts.

II.5 World Wide Web browsers

The goal of the project is to develop a Web-centric application. This application should be able to execute on a Web Browser. But at this time the application does work only with the Netscape Communicator browser (version 4.7 and upper versions).

We have been faced with some compatibility problems with this browser, and more with Microsoft Internet Explorer browser. In fact, to execute the application within the Explorer browser, we need to use a lot of artifices in order to allow some *Java* components (such as JDBC, or the Swing API) of JDK 1.2.1 to work with this web browser. But for Netscape Communicator, we only need to add a file of *Java* classes (*"Swing.jar"* available on http://www.java.sun.com/products/) to the *Program Files\Netscape\Communicator\Program\ Java\Classes* directory to enable swing so as to be able to execute the application.

III. Actual Implementation

At this point we will discuss the implementation of the classes that allow the dynamic generation of the two charts, that is to say the *ScatterChart* and the *BarChart*.

The *Datum*, the *Semantic* and the *Representation*[40] classes are the same for both of charts. There is one generic class, the *Chart* class. This one is declared as an **abstract**[41] class, and includes the methods that will be the same for all the possible charts. It will be so for the method that initialises a chart, by displaying all the necessary legends within the *VRML* world. The *Chart* class includes also the signature of other methods (**abstract** methods). These methods will be different for each chart. It will be the case for the method that adds the points (i.e. the *VRML* objects) into the virtual world. In fact, this method will be different if we are confronted to the *ScatterChart*, or to the *BarChart*.

The two specific classes, *ScatterChart* and *BarChart*, will **extend** the *Chart* class. They will provide a particular implementation for the **abstract** methods of the *Chart* class, and will inherit the implementation of the other methods.

Now we need to give the user the possibility of choosing the chart, and all the parameters with regard to this chart. This will be achieved via the Graphical User Interface (the five different frames presented in the chapter three related to the design of the application).

As the user enters his choices, we generate a vector of semantic objects that will materialize the user's choices. This vector, together with the title of the chart, will be passed to the constructor[42] of either the *ScatterChart* or the *BarChart* class. After the multiple accesses to the database, we will get a **resultSet**[43] and the *Interface* class will pass each row of this **resultSet** to the instance previously created of either the *ScatterChart* or the *BarChart* class.

The *Interface* class will also enable the display of a frame with the legend. This frame should always be available to the user, especially if there are many dimensions represented on the chart.

---

[40] All the classes we introduce here will be explained later in this chapter. You can also refer to the chapter concerning the design of the application where the classes and their content are defined precisely. The appendix c contains the java implementation of the main classes.

[41] In Object-Oriented programming, there are cases in which it is useful to define classes for which the programmer never intends to instantiate any objects. Such classes are called *abstract* classes. They are used as superclasses in inheritance situations. The sole purpose of an abstract class is to provide an appropriate superclass from which other classes may inherit interface and/or implementation. Classes from which objects can be instantiated are called *concrete* classes.

[42] When an object is created, its members can be initialised by a *constructor* method. A constructor is a method with the same name as the class (including case sensitivity). The programmer provides the constructor which is invoked automatically each time an object of the class is instantiated. [Deitel&1999]

[43] A **ResultSet** is a set that contains the result(s) of a query to a specified database.

III.1 The ScatterChart Class

This class creates *VRML* Objects according to its semantics field. Whenever the *Interface* Class calls the method *addRow* of this class, it will progressively add objects to the *VRML* world.

First, the semantics field enables the class to give a caption to the different axes. For example, just consider Figure 38. ScatterChart is the chart title. The x- and y-axes both have a legend along their axis: Arithmetic_2000 and El_Algebra_2000 respectively. The z-axis also has one but we cannot see it on this screenshot.

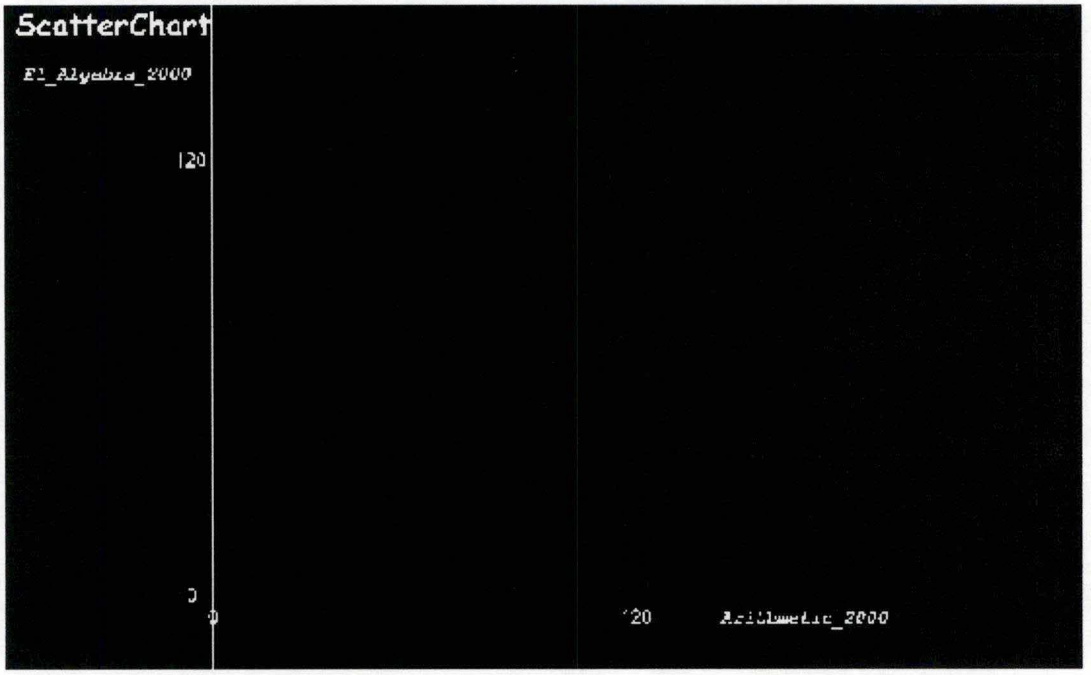

Figure 38: The legend on the ScatterChart axes

After this we only need to add the different *VRML* Objects. Each object here will be a particular shape with a particular size, colour (or texture), transparency, and at a particular 3D position.

In our example in Figure 39, there are 512 students represented on the chart. We can also see the navigation tool at the bottom. If we look at the status bar, we can see accurate information displayed about a particular student (the current focus).

There is also a legend displayed together with the chart. The user has always to know what data is in front of him. We can see an example of legend in Figure 40. This example illustrates the use of six dimensions at a time.

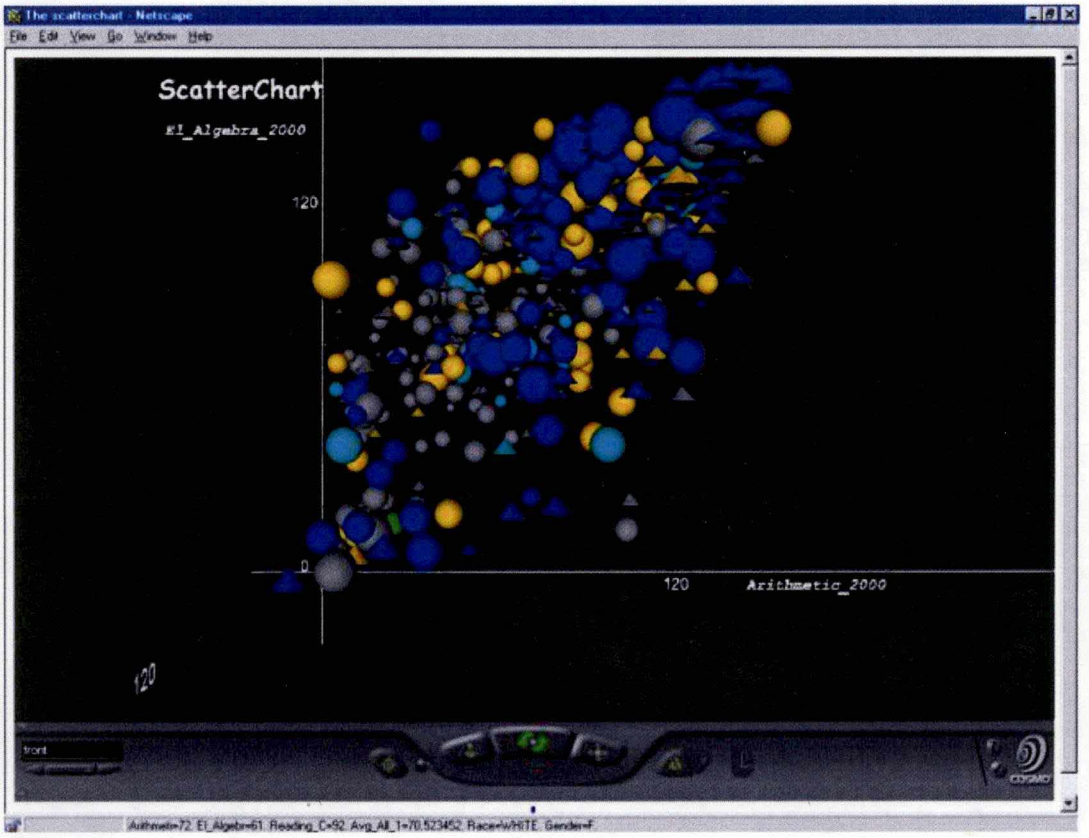

Figure 39: The ScatterChart: front view

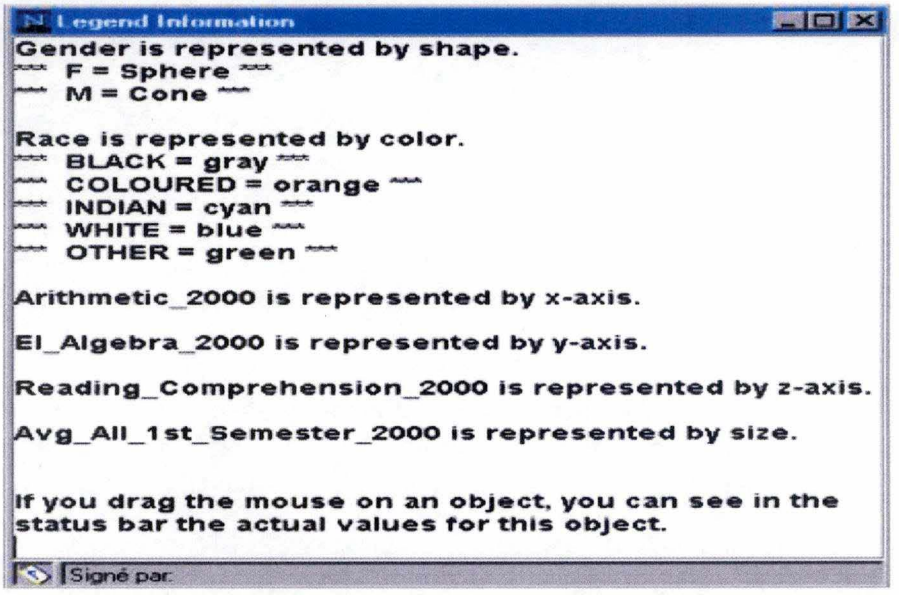

Figure 40: The ScatterChart legend

The next figure (Figure 41) illustrates the same chart but from another viewpoint. Here we can see the progression with regard to the z-axis (Reading_Comprehension_2000). At the left bottom, there are six default viewpoints, but the user can navigate in the world as he wants.

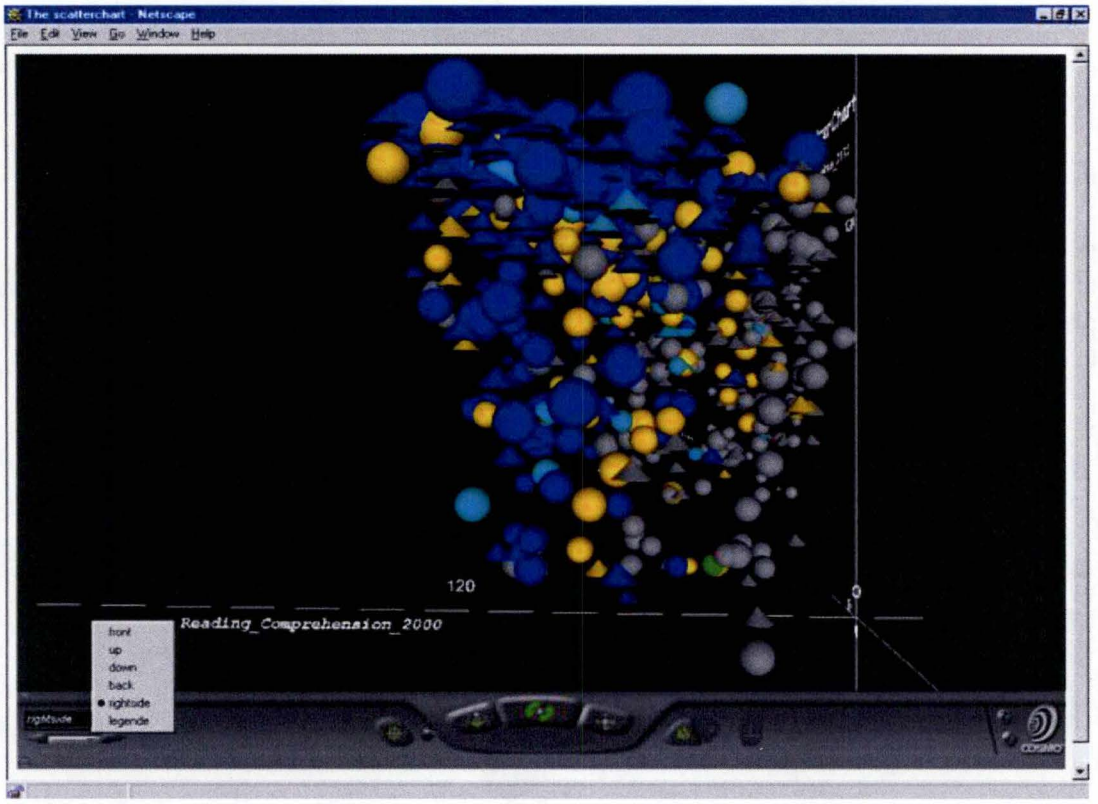

Figure 41: The ScatterChart: side view

III.2 The BarChart Class

Here also the semantics field will be used to give the axes a legend, and to give the chart a title. But the way to add *VRML* Objects differs.

First, all the objects are bars (or boxes). They all have the same width and depth. Only their 2D position (x- and z-axis, i.e. the non-quantitative axes) and their height (y-axis) differ. If we take into consideration only the semantic properties of this chart, the colour also should be the same for all the bars. But our implementation provides a different colour with regard to the position on the x-axis. It is obviously better for the user if only the bars of a same row have the same colour. So he could examine the chart most efficiently and in an easier way.

The next figure (Figure 42) illustrates a 3D Bar Chart with all the same colour bars.

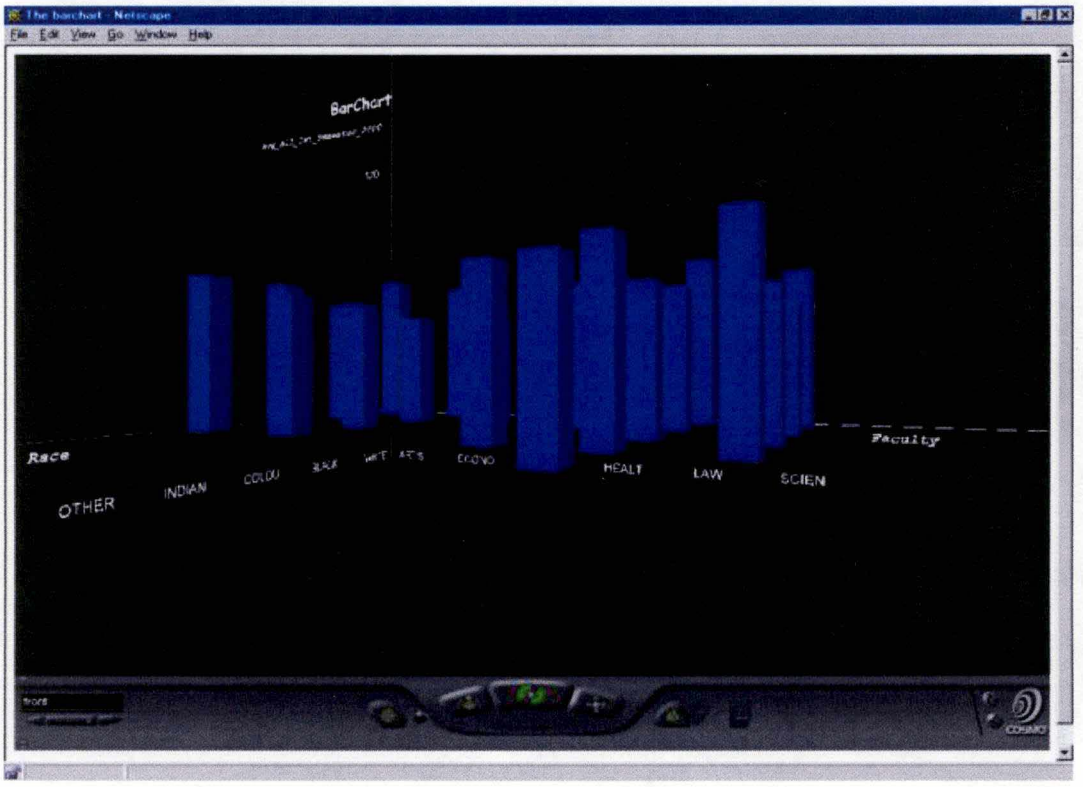

Figure 42: The BarChart: one colour

We cannot really see clearly all the bars. The faculty is represented on the x-axis; the race on the z-axis and the y-axis is used to convey the average of all the students for a given race and a given faculty. If we want to view the accurate average for a bar, we just need to drag the mouse on it. We will also have at our disposal the number of students represented by this bar.

In the chart represented in Figure 43 we have used a different colour for each faculty. We can clearly distinguish a bar representing a particular faculty with another one representing another faculty. The z-axis is used to convey the gender variable: the bars at the back represent the female gender, and the others represent

the male gender. Of course, the user can always move around the world by using either the predefined viewpoints or the dashboard of the browser.

In the ScatterChart each object represents a student, but here, most often, it represents a group of students. So the queries to access the database are different. We will often have to calculate an average.

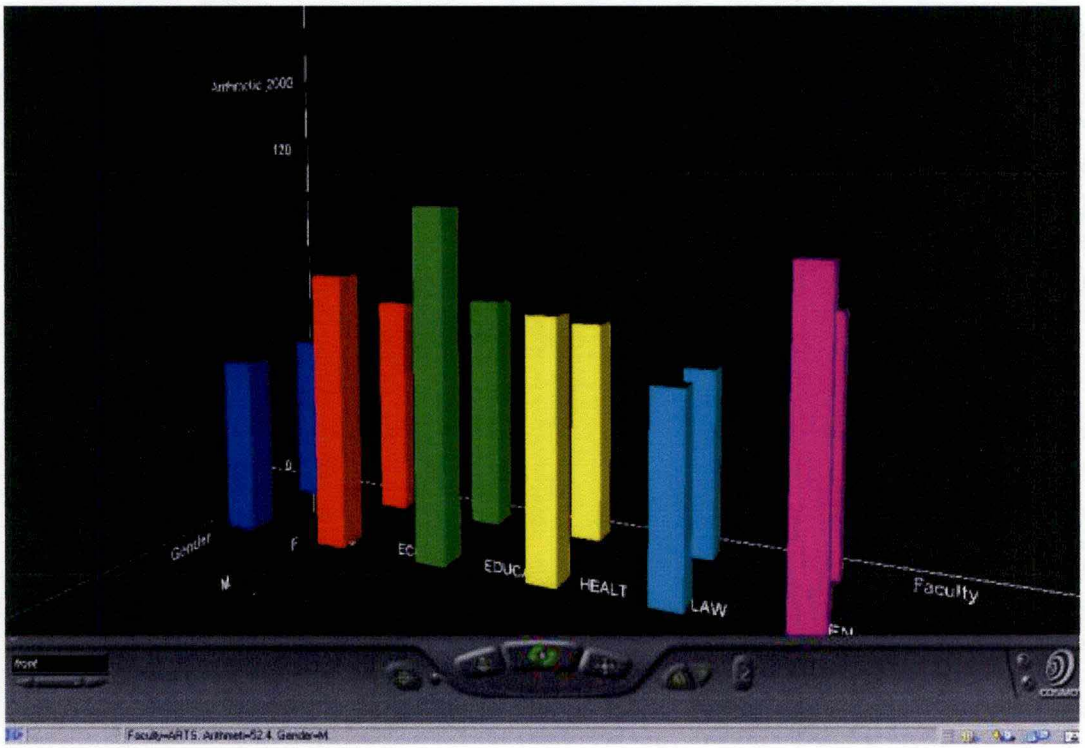

Figure 43: The BarChart: different colours

IV. Conclusion

We have shown that the technology used was sufficient to develop the *Visualization Application*. And the method used to build *VRML* Objects using *Java* is not difficult, but quite long.

Concerning the future possible researches, there is a need to try and extend the application to other charts, and of course to improve this one. We have to give the user flexibility in term of different possible charts. One should also try to give the application a full interactivity, notably in the direct manipulation of the charts that are displayed in virtual worlds. Up to now the only direct manipulation we offer to the user is provided by the navigation tool, and this tool does not allow the user to modify the virtual world. He can just navigate as he wants throughout the world.

But these first results are rather promising and further researches should proceed in this direction.

Chapter Five

Task analysis[44]

I. Structure of the task

This point follows the **TKS Model**[45] developed by P. Johnson in [Johnson92].

Main goal : visualize a large data set that answers to some constraints with the aim of observing comparisons, trends. User can customise the chosen chart, thanks to the User Interface and manipulate it (zoom, pan, rotate, and so on), thanks to the navigation tool available in the *virtual world*. The aim is to improve the current 2D visualization tools by providing a better perception of the data so that the user can do a better analysis and take more accurate decisions.

Sub-goals : there are two main sub-goals : the choice and customisation of a chart on the one hand, and the handling of the displayed chart on the other hand.

Plan :

1. The choice and customisation of a chart (*sub-goal*)
   - 1.1 Selection of a 3D Chart (only two possibilities: the BarChart and the ScatterChart). (*procedure*)
     - 1.1.1 chart selection : the user chooses between the two possibilities according to his needs. (*action*)
     - 1.1.2 task continuation : if the user wants to go on, he presses the "next" button and accomplishes the next sub-sub-goal (cf. 1.2). (*action*)

Inside the next sub-sub-goals (1.2, 1.3 and 1.4) and inside the sub-sub-sub-sub goal (1.4.1.2), the user can check the "help" menu (if he is not yet used to the application). He can also cancel it to return to the previous one (respectively 1.1, 1.2, 1.3, and 1.4.1.1) and modify the choices he previously did.

   - 1.2 Particularisation of the chart by choosing a title and selecting a specific subset (some columns) of the database with regard to the dimensions the chart can offer. This subset contains the data the user wants to visualize. (*sub-sub-goal*). There is here no order between the *procedures* 1.2.1 and 1.2.2.
     - 1.2.1 choice of a title for the chart (*procedure*)
     - 1.2.2 choice of columns (*procedure*)
       - 1.2.2.1 columns selection (*action*)

[44] The Task Analysis was achieved according to the model available in [Bodartb1998].

[45] TKS is an acronym for Task Knowledge Structure.

1.2.2.2 columns deletion (*action*)

1.2.3 task continuation : to accomplish the next sub-sub-goal (cf. 1.3). (*action*) But if the user's selection is not authorised, he must repeat the 1.2.2 procedure.

1.3 Constraints (if necessary) on the subset (*sub-sub-goal*)

1.3.1 constraints registration : the user can specify a constraint **for each column** selected at the previous point. (*sub-sub-sub-goal*)

1.3.1.1 registration for nominal columns (*procedure*)

1.3.1.1.1 data selection (*action*)

1.3.1.1.2 data deletion (*action*)

1.3.1.2 registration for quantitative columns (*procedure*)

1.3.1.2.1 choice of a comparison symbol (*action*)

1.3.1.2.2 padding of a field with an integer (*action*)

1.3.1.3 task continuation : to accomplish the next sub-sub-goal (cf. 1.4). But if the constraints are too restrictive so that no element of the database verifies these constraints, the user must repeat the 1.3.1 sub-sub-sub-goal. (*procedure*)

1.4 Association of **each column** (the columns selected previously) with a concrete dimension that will be represented into the virtual chart. (*sub-sub-goal*)

1.4.1 dimension association (*sub-sub-sub-goal*)

1.4.1.1 choice of a dimension. (*procedure*) This procedure is followed by the 1.4.1.2 sub-sub-sub-sub-goal if and only if the dimension equals to colour, shape, texture or transparency.

1.4.1.2 custom association (this task is only executed after the 1.4.1.1 procedure). (*sub-sub-sub-sub-goal*)

1.4.1.2.1 association between a value and a representation of the chosen dimension (*procedure*) **for each different value** of the column

1.4.1.2.1.1 choice of a value (*action*)

1.4.1.2.1.2 choice of a representation (*action*)

1.4.1.2.1.3 association (*action*)

1.4.1.2.2 value disassociation of the last association (*procedure*)

1.4.1.2.3 validate custom association when all the values have been associated with a representation (*procedure*)

1.4.2 dimension disassociation (*procedure*)

1.4.3 task continuation : display the chart when all the associations have been performed. (*procedure*)

2. The handling of the displayed chart (*sub-goal*)

2.1 Manipulation of the virtual world (*sub-sub-goal*)

2.1.X Use of all the possibilities offered by the navigation tool : moving (walking or flying) around the world, zooming, rotating, panning, sliding, tilting, seeking for objects, and reaching a predefined viewpoint. You can also use the Undo Move and Redo Move buttons to retrace your steps. (*different procedures*)

As we can observe it in the sub-sub-goal 2.1, the manipulation of the virtual chart concerns only the manipulations offered via the navigation tool. There is no interface embedded in the virtual world that allows him to directly modify the chart. If he wants some modifications for a chart (e.g. use the colour dimension instead of texture), he needs to begin the task from scratch. So, if any future research could provide the user with such an interface, it would be a major improvement for the *Visualization Application.*

II. User's stereotype

We described here the stereotype of the user, who will use (or could use) the *Visualization Application* to analyse the *UPE* students' data. The user wants to examine the students' results coming from a large data set. His aim is to observe comparisons, trends and so on, and to be able to draw some conclusions (or explanations) from the charts. He has decided to analyse this data set thanks to the 3D visualization in virtual environments.

- **Experience of the abstract task** : he is used with data analysis (through other means than the 3D visualization) and knows perfectly the contents of the database. He is so familiar with the abstract task since he already performed it. The major change consists of the way of visualizing the data. These ones are indeed displayed in three-dimensional virtual worlds whereas the user is used to a two-dimensional display.

  The experience of the task may so be described as medium or even high.

- **Experience of an Information System** : he does not need to have any particular competence to handle and use the interface. At *UPE*, the user possesses greatly the necessary experience thanks to their previous projects.

  The required experience of an information system is elementary because the application is quite intuitive.

- **Motivation to use the system** : if the user is interested in 3D visualization, his motivation will be high, because the interface and the handling of the chart are intuitive and easy. The possibilities of interaction will affect encouragingly the motivation. The limitations of the current visualization tools and the greater effectiveness of this one are the main motivations.

- **Experience of a complex mean of interaction** : the user is used to the 2D manipulations but not really with the 3D, and so he has no experience in the 3D interaction.
  There are two means of interaction. Before the display of the chart, the user interacts with the application via several intuitive interfaces. Once the chart is displayed, the user interacts thanks to the navigation tool in the virtual world. So he makes use of a Visual User Interface (VUI). It's not very easy at the first time to travel within this world but with a little time, the user can easily get through this problem. The experience is so not crucial, it will just make the task easier.

- **Knowledge of a similar concrete task** : he knows concepts that refer to the visualization of a chart (the notion of dimension, of nominal and quantitative data, and the like). Therefore, he can quite easily perform the steps before the display of the chart. However he does not know necessarily what is a BarChart and a ScatterChart. Also, he does not know what are all the possibilities of the 3D visualization and how to make good use of it, because the use of a VUI to interact with a 3D chart is not very usual.

  Therefore we can consider that the user has only a partial knowledge of a similar concrete task. However he can learn and get used to this by spending time on the application.

Now, let's conclude this section. The abstract task is mastered by the user, and he is quite experimented concerning the data analysis. The only difference is that now he has to deal with this task in a virtual world. He already owns other means to realise his task (like the 2D tools).

So, the application will have to make easy his analysis without making the concrete task harder. The main asset being the use of a VUI to interact with a 3D chart. This asset won't be immediate, a period of adaptation will be perhaps necessary.

The application can work with other databases. So different users from the ones described just above could exist. Their stereotype will be more or less the same. But their experience of the abstract task and their knowledge of a similar concrete task won't be necessarily so high. The motivation will increase with the use of the tool,

when he will find out all the possibilities and when he will be able to complete easily what he wants to do.

III. Descriptive parameters of the task

1. **Required knowledge :** the user does not need to have any particular competence to handle and use the interface, but he has to have some experience of the abstract task. The tool will make easy the analysis of data but won't perform this for him. So he must be able to do an analysis from a displayed chart. However he will be able to learn and to get used to 3D visualization and manipulation by spending time on the application. The requirements are so moderate.

2. **Productivity :** it depends on the user. Productivity will be considerate like mean. At *UPE*, if this tool replaces the 2D visualization tools, the productivity could be high.

3. **Objective environment :** allows the visualization of some multi-dimensional abstract data to make easy their analysis.

4. **Reproducibility of the environment :** not applicable because there is no objective environment to reproduce (abstract data are not reproducible by definition).

5. **Task organisation :** the user can go to the next window or return to the previous one to correct his errors. Inside a window, he fills in it in the order he wants. The inflexibility is so moderate.

6. **Importance of the task :** the application is very interesting to analyse a data set, but it is not vital because there are other visualization tools that can be applied for the analysis.

7. **Complexity of the task :** the complexity to display a chart is low or moderate, while the handling and the analysis of a chart may be high.

Conclusion : as the objective environment is abstract and no reproducible, the cognitive distance is automatically high. Because the user works in a world that is not familiar. The validation of the application is so more complicated. The application, to be useful, will have to be particularly intuitive, easy to use, and provide many feedback and also *inter-referential inputs* and *outputs*. The progression of the task will have to be clear for the user. It is very important considering the length of the task. Thus, he will be able to interrupt it and will easily rectify his intention errors.

IV. Work's context

1. **Physical environment :**
   Hardware : the use of Java and VRML causes that there is a need of a good processor (e.g. more than 400Mhz). Of course due to the 3D visualization, a good screen resolution is also necessary.

   Software : Netscape Communicator 4.7 (and upper versions), Cosmo Player, Java, VRML (External Authoring Interface), and tools that allow you to manipulate databases via the JDBC component of Java. These components will be presented in detail in the Appendix B *How to launch the Visualization Application.*

2. **Mono- or multi-processing assignment :** it's a mono-processing assignment. The user does not need to do another task. He has just to follow the progression proposed by the Graphical User Interface. Since the task is long and can last a few minutes, the user will have of course the opportunity to interrupt it. He will be able to continue it further without any problem. The analysis of the chart can also be done afterwards because the displayed chart reminds of what has been chosen in the GUI. As long as you keep open the virtual world (embedded in the VRML browser), the chart stays available.

3. **Modes of a task execution :** the user has the control of the dialog. Inside a frame, the user fills in it as he wants, without any particular order. He can return to the previous frame or, when he has finished to fill in it, go to the next. He can also work on different visualizations, going from one to the other to make comparisons.

V. Utility and Usability criteria

1. **Ease of learning** : this is not the most important criterion. The time of learning should be nearly non-existent to be able to display a chart. The application should take advantage of the experience that the user has of 2D visualization tools. However, he should spend a little time using the application to get used to the handling of charts in a virtual environment. This necessary time is indeed not a problem because, as we have already told it, at *UPE*, if this tool replaces the 2D visualization tools, the productivity could be high.

2. **Efficiency of use** : it is not a crucial point. However, like for all application, you have not to make lose the user's patience. So the different steps before the display of the chart should not take a long time, in order to allow the user to remember the choices he has made previously.

3. **Memorability**: it is essential considering that the user will often work with the application.

4. **Error frequency and severity**: the execution errors must be nearly nil and never disastrous. The intention errors must be easily corrected by allowing the user to detect and undo them rapidly.

5. **Subjective satisfaction** : it is only on condition that the user is satisfied that he will use the application. It is the most important criterion seeing that there are other 2D visualization tools that can be applied to the analysis of data.

Now, we are going to consider the compliance of the proposed interface. This interface is intuitive and provides feedback to make easy the learning and the efficiency of use. The execution errors are treated and the user is immediately and accurately notified. The intention errors can be easily corrected inside each frame and the user can also directly return to the previous frames. He can detect some of them by observing the frames but unfortunately for others he will have to wait the display of the chart. With the experience, the chart generation, the handling of this one, and the analysis of the data present in the chart will be faster. His subjective satisfaction will automatically increase.

CHAPTER SIX

Functional Architecture

I. Introduction

This part presents the functional architecture (derived from the *Task Analysis* chapter) of the *Visualization Application*'s interface we have developed at the University of Port Elizabeth. We will show how to decompose the different functionalities of the application, how do the functions call each other.

We can consider all the application as an interactive task where some control objects pilot the execution of the task and coordinate the exchanges between the presentation and the application services. The presentation component corresponds to the User Interface (cf. chapter three *Design of the Visualization Application*), where he can enter and receive the pertinent data. The application services are the services we want to offer to the user.

II. Architecture

For the architecture we are going to use the Chaining Graph (CG) notion [Bodartb1998]. A CG represents the chain of the different main *functions* that constitute the interactive task. Each function corresponds to a service we offer to the application user.

In order to be able to build the CG we have to take a look at the User Interface. This interface is composed of two presentation units (PU). The PU's are the main components of the interface that the user has to deal with.

The first one (PU_1) includes four "windows" (W_{11}, W_{12}, W_{13} and W_{14}) and represents the sequence of different sub-tasks associated with the main interactive task. At the beginning, we had the intention of putting them together in one tab pane. But we have not done it because the user must fill in these "windows" in a particular order. We have discussed about that in the chapter *Design of the Visualization Application* (section VI). So it was logical to bring them together in a same presentation unit. Let's analyse each of these windows one by one.

1. W_{11} : as we consider a database chosen by default, the first thing a user has to do, is to select a 3D Chart (till now only two possibilities: the BarChart and the ScatterChart).

2. W_{12} : with regard to the dimensions the chart can offer, the user will choose a specific subset (some columns) of the database. This subset contains the data he wants to visualize. (first filter on the database)

3. W_{13} : the user can also decide to visualize only a part of the subset (cf. point 2). In fact, he could want to visualize only the male students (e.g. in the case of the data set from UPE). In this case he has to enter a specific constraint on the selected column. (second possible filter on the database)

4. W_{14} : and eventually, he has to associate each dimension (the columns selected previously) of the database with a concrete dimension that will be represented on the chart. The association is a crucial point of the application and so, we will give feedback and warnings to the user; if necessary.
   An association can necessitate a particular treatment from the user (cf. PU_2).

The second presentation unit (PU_2) consists of only one window (W_2). Each time an association in W_{14} necessitates a particular treatment, we will need an instance of W_2. Here the user can associate to each different database value (of the column relative to the association) a proposed value relative to the concrete dimension chosen (cf. point 4). W_2 can so be "called" several times from W_{14}, that is the main reason for which we have put it in a different presentation unit.

So now, we can propose different aggregate functions. The CG (cf. the figure proposed at the end of this chapter, Figure 44) takes into consideration all these functions and their chain.

We find there six functions:

1. **chart selection**: it needs a message representing the selected chart. Here the user chooses between some possibilities, so there is no possible ambiguity.

2. **columns selection**: the function takes in entry a title for the chart, and all the columns selected from the user. Either the selection is valid and he can switch to the next point, or an error occurs during the execution of the function (e.g. the number of columns selected could be too high for the chart).

3. **constraints register**: the function analyses the entered constraints and returns an error message if they are not valid. The user can specify a constraint for each column selected at the previous point. He can also decide to keep all the data and to enter no constraints.

4. **dimension association**: after going through the previous points, he will associate each column (cf. point 2) with a dimension. At this stage, a lot of errors could occur (e.g. incompatible associations, redundant associations, and so forth), if we are not watchful during the implementation of the application.

5. **visualization**: this is the final and main function. It takes in entry all the pertinent messages produced from the other functions. The only possible result here is a visualization of the data that interest the user in the selected chart.

6. **custom association**: around this function there is a particular structure in the CG. There is a repetition of the messages and of the function. In fact, as explained before, the user could need several (from 0 to i, the number of dimensions) instances of W_2. In each instance, the function needs in entry a valid association (between a column and a dimension) and b messages, where b is the number of the different possible values of a column. The b

the colour dimension (association *gender-colour*). The b value is two (*male* and *female*). So the two different messages could be *blue* and *pink*.

We can also notice that some functions generate several messages. In this case, there is an arc joining the arrows. It means that the function will generate only one message (e.g. in the window W_{12} the function column selection generates either the "error" or the "valid selection" message). At the other hand, each function needs all the entry messages represented in the CG.

In conclusion, PU_1, PU_2 and the different windows constitute the presentation component of the architecture. The functions represented with rectangles form the functional component. They correspond to the different public methods of the interactive task. Of course, these public methods can call others one (private methods) not represented in the CG. For example, the visualization function that will produce the requested result, needs many private methods. With regard to the manipulations from the user thanks to the interface, a conversation component will manage the dialog between the different windows.

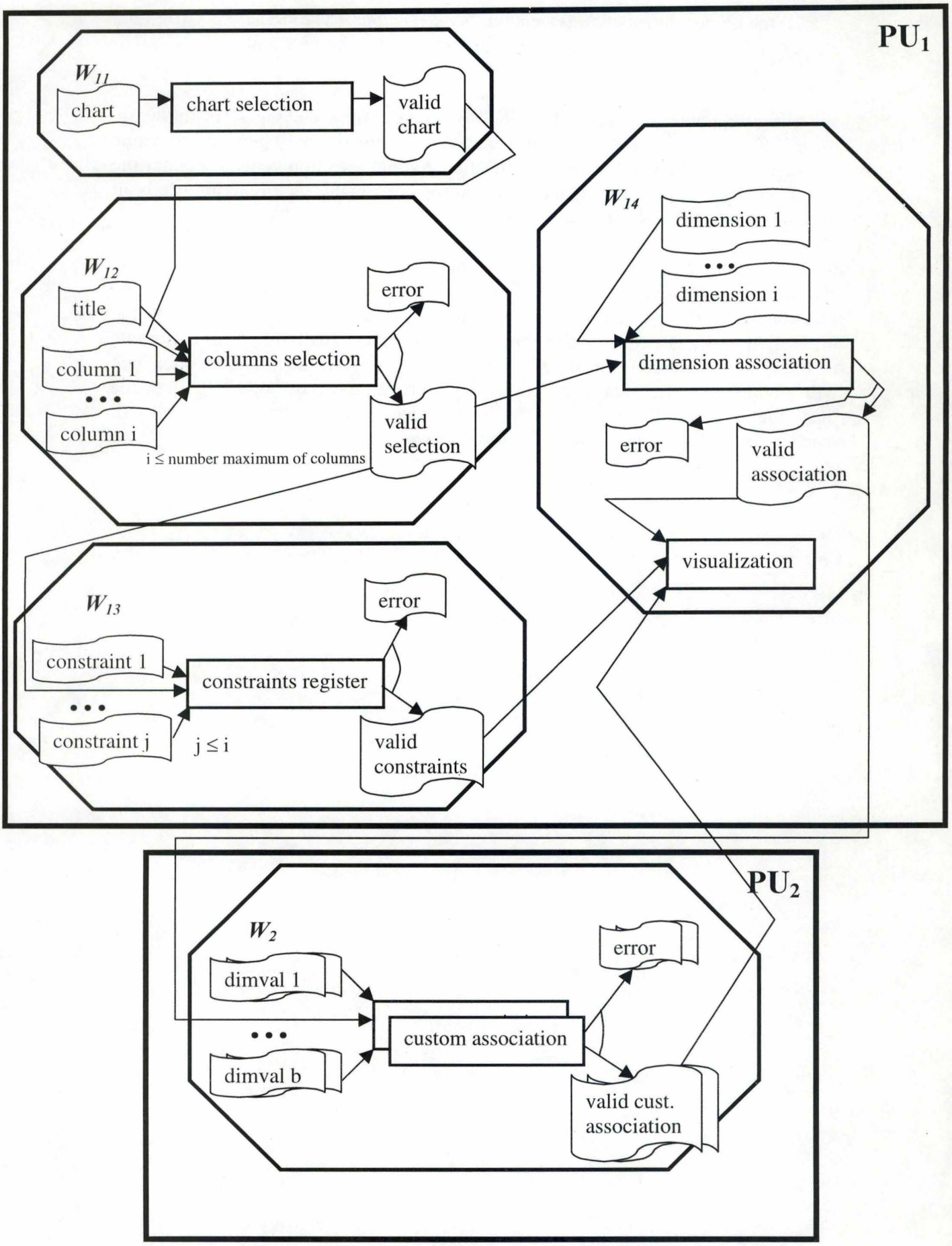

Table 3: Chaining Graph

Conclusion

Conclusion

I. Achievements

We have realised the four major steps identified at the beginning of the thesis. In this conclusion, we highlight the chapters that achieve these steps.

In the first chapter, *Semantic Analysis of Data Visualization*, we have identified a **suitable data set** as an application area for the thesis; this is the Aptitude Test results from the **Accuplacer** system. The presentation of UPE's case has also been done. Next, we have presented and analysed the **semantic properties** (restricted to the goals of the *Visualization Application*) of the data and the charts. To build dynamically a chart, we have just needed to deepen the semantic property with reference to the data type. We have applied this to the **Accuplacer** system's data set. Finally, we have identified the different suitable charts for the application and have decided to privilege two of them : the ScatterChart and the BarChart.

In the second chapter, *Assessment of Interactive Visualization*, we assess the use of the Interactive Visualization for large data sets. We try to develop the hypothesis that interactive 3D techniques can be useful to help the user understand, analyse and filter such data sets. This chapter also contains a utility and usability presentation and analysis with regard to the application we want to create. After that, we have brought you into contact with the concepts of Interactivity and Graphical Visualization. Then we have seen how interactivity and graphical visualization can assist us in reaching one of our goals, namely the Interactive Visualization of large data sets. Next, we have justified the choice of three-dimensional representation, by comparison with two-dimensional representation. Eventually, we have introduced and define the concept of *Virtual Environment* (VE). The main advantage of the VE being that he reduces psychological distance by representing task domains in a more realistic manner and facilitating more natural, multi-modal interaction. We have also presented the benefits of the visualization in such environments, and the common problems encountered in this field. With this chapter we have begun the critical **assessment** of the current (and possibly future) ability of the *Virtual Reality* to allow such an Interactive Information Visualization.

Then, in *Design of the Visualization Application,* we have given the specifications and the architecture for the part of the application concerning the generation of different charts, and we have proposed a prototype for the User Interface (UI). We have first specified an architecture that is general for any chart. Then, by using the principles of inheritance, we were able to propose a specific architecture for a given chart. Therefore, even if we have abandoned the use of *Java Beans* technology, nevertheless the application is fairly reusable. We have then implemented two charts : the *ScatterChart* and the *3D BarChart.* The implementation is quite different but each one extends and uses the implementation provided by the generic chart class. Then, we have proposed an interactions' scheme that features and explains the different interactions between the main components of the architecture. You have also been able to see an example of how this architecture can be extended

to include other charts. Afterwards, we have exposed what we've done concerning the conception of the UI and have discussed the design of this one. Eventually, we have summarised all this by putting forward a global architecture of the *Visualization Application* in its entirety, including the interface components, the database accesses, and so on.

In the *Implementation* chapter, we have presented the real implementation of the application. Also, the **Implementation tools** we have used to achieve this goal have been discussed, notably in a **requirement** perspective. We have shown that the technologies used were sufficient to develop the application. Examples of the actual implementation are given as well.

Eventually, in the chapter *Task Analysis*, we study the user's stereotype, the work's context, the task's structure, and the Utility and Usability criteria. This is another point of the *critical assessment*. From this task analysis, we have then derived a *Functional Architecture*.

II. Problems encountered

We have encountered the first problem during the implementation of the chart classes. We cannot do in VRML all what we would want. For example, we had considered colour and texture as two different dimensions. But in VRML, we cannot visualize both at the same time. We thus have one dimension less.

> "If the texture is a color image, the texture's colors override any color specified in a Color node or in the diffuseColor field of a Material node". [VRML2000]

We have also encountered problems of compatibility:

- Between Netscape and Java Swing Components.
- Between Internet Explorer and Swing, and also with the JDBC-ODBC API.

We were able to solve the problem between Netscape and Java Swing, but by lack of time we had to give up the other compatibility problems.

III. Future research

First of all, it should be emphasized that one can use the application only as an Applet. The entry-point class has to be an applet that we will first run in a Web browser. This limitation results from the Cosmo Player. Perhaps one could find a tool other than Cosmo Player which would enable to structure this differently.

We have only implemented the Scatter Chart and the 3D BarChart. One may thus extend also the *Visualization Application* and implement the Scatter Ribbon Chart, and probably also the 3D Tree and the Temporal Star because their semantics is quite different from the two previous ones. The Graphical User Interface is not finished either, and needs to be improved in terms of usability and usefulness requirements.

Another main improvement concerns the interactivity we want to give to the user. If we wish to increase the user's motivation, we need to provide him with a flexible application. Because the major problem of the current application is that the user can only visualize the chart when the task is completed. Once the chart is displayed, he will certainly want to modify directly and interactively certain characteristics of the chart. Thus, as proposed earlier, we need to offer this possibility to the user.

A future research could also try to complete the implementation by developing a Web application. To allow the user to choose the data set he wants to visualise would be another appreciable improvement. The data set should not necessarily come from a database (like Microsoft Access) but the user should also have the possibility to select this data set in a more flexible way (such as the selection of data from a table sheet). The application's potential would be then much higher.

REFERENCES

References

- [Beier2000] Beier K.-P., *Virtual Reality: A Short Introduction.* University of Michigan, *Virtual Reality Laboratory* at the College of Engineering. February 2000 (from http://www-vrl.umich.edu/intro/index.html).

- [Bodarta1998] Bodart, François (1998), *III.3 Critères ergonomiques de conception d'une interface.* Cours introductif à la conception des Interfaces Homme-Machine.

- [Bodartb1998] Bodart F., *Chapitre IV: l'analyse de la tâche. Cours introductif à la conception des Interfaces Homme-Machine.* FUNDP, 1998.

- [Brutzman98] Brutzman D., *The Virtual Reality Modelling Language and Java.* Communications of the ACM 41(6): 57-64, June 1998.

- [Bryson95] Bryson S., *Virtual Reality Applications.* London: Academic Press, 1995.

- [Bryson96] Bryson S., *Virtual Reality in Scientific Visualization.* Communications of the ACM 39(5): 62-71, May 1996.

- [Card1999] Card, Stuart K. (1999) *Readings in Information Visualization: Using vision to think/write.* Edited by Stuart K. Card, Jock D. Mackinlay and Ben Shneiderman, Morgan Kaufman.

- [Carr&93] Carr K.T. and England R.D., *Virtual Reality International `93: third annual conference.* London 1993. Proceedings. London: Meckler, pp. 24-33.

- [Darville&2000] Christelle Darville and Stéphane Van Espen, *Study of the Virtual Reality Technology used in the Visualization of Business Information.* MSc

Dissertation, University of Namur, September 2000.

- [Deitel&1999] Deitel, H.M. and Deitel, P.J. (1999) *Java: How to Program*, Third Edition. Prentice-Hall.

- [Ellis1993] Ellis S.R., *Pictorial Communication in virtual and real environments*. London: Taylor and Francis, 1993.

- [Gigante1993] Gigante M.A., *Virtual Reality Systems*. London: Academic Press, 1993.

- [Hang&1998] Hang Ieong C., Oon Tan C. and Ming Kwong Ti A., *Virtual Campus: The University of New South Wales - Chapter 3*, May 1998, http://www.cse.unsw.edu.au/~lambert/vrml/bdmx/Report/Report.html.

- [Hubbold&93] Hubbold R., Murta A., West A. and Howard T., *Design issues for virtual reality systems*. Paper presented at First Eurographics Workshop on Virtual Environments, Barcelona, September 1993.

- [Johnson92] Johnson, P. (1992), *Human Computer Interaction*, Mac Graw-Hill

- [Kaur1999] Kaur K.D., Sutcliffe A. and Maiden N., *A design advice tool presenting usability guidance for virtual environments.* Paper presented at The Workshop on User Centred Design and Implementation of Virtual Environments. York, UK, September 1999.

- [Loeffler&1994] Loeffler C.E. and Anderson T. (1994). *The Virtual Reality Casebook.*

- [Loomis93] Loomis J.M. (1993). *Symposium on research frontiers in virtual reality, San Jose, California 1993. Proceedings*, Los Alamitos: IEEE Computer Society Press, pp. 54-57.

- [Lycos2000] LYCOS Inc., *"Virtual Reality - Tech Glossary"*, http://webopedia.lycos.com/TERM/v/virtual_reality.html, 2000

- [Miller1956] Miller G., *The Magical Number Seven, Plus or Minus Two : Some Limits on our Capacity for Processing Information*, Psychological Review 63, n°2 (1956): pp. 81-97.

- [Nielsen93] Nielsen Jakob, *Usability Engineering*. San Diego: Academic Press, 1993.

- [Nielsen98a] Nielsen Jakob, *What is Usability?*, http://www.zdnet.com/devhead/stories/articles, September 14, 1998.

- [Nielsen98b] Nielsen Jakob, 2D is Better Than 3D, http://www.useit.com/alertbox/981115.html, November 15, 1998.

- [Noirhomme00] Noirhomme-Fraiture Monique, *Multimedia Support for Complex Multidimensional Data Mining*, Congrès MDM (KDD 2000 Workshop), 2000.

- [Pausch1993] Pausch R., *Three views of virtual reality: An overview*. Computer 26(2): 79-81, February 1993.

- [Preece1994] Preece, Jenny, *Human-Computer Interaction*, Addison-Wesley publishing company, 1994.

- [Robertson&93] Robertson G.G., Card S.K. and Mackinlay J.D., *Three views of virtual reality: non-immersive virtual reality*. Computer 26(2): 81-83, February 1993.

- [Rushton&93] Rushton S. and Wann J., *Virtual Reality International '93: third annual conference*. London 1993. Proceedings. London: Meckler, pp. 43-55.

- [Shackel91] Shackel B. and Richardson S., *Human Factors for Informatics Usability,* Cambridge University Press 1991.

- [Shneiderman92] Shneiderman B., *Designing the user interface: strategies for effective human-computer interaction,* Second edition. Massachusetts: Addison Wesley.

- [Singh&1996] Singh G., Feiner S. K., and Thalmann D., *Virtual Reality: Software and Technology.* Communications of the ACM 39(5): 35-36, May 1996.

- [Singhal&98] Singhal S. and Nguyen B., *The Java Factor.* Communications of the ACM 41(6): 34-37, June 1998.

- [Strickland&97] Strickland D., Hodges L., North M., and Weghorst S., *Overcoming phobias by Virtual Exposure.* Communications of the ACM 40(8): 34-39, August 1997.

- [Strickland97] Strickland D., *Virtual Reality and Health Care.* Communications of the ACM 40(8): 32-33, August 1997.

- [Tebbut1999] Tebbut, David. (1999) *Desktop Evolution.* URL: http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/582

- [Travis&94] Travis D., Watson T. and Atyeo M., *Interacting with virtual environments.* Chichester, UK: John Wiley and Sons, 1994.

- [Tufte1990] Tufte, E. (1990) *Envisioning Information.* Cheshire, CT: Graphics Press.

- [Tyma1998] Tyma P., *Why are we using JAVA AGAIN?.* Communications of the ACM 41(6): 38-42, June 1998.

- [Vesale2001] Grosjean C., *The Cognitive Walkthrough*, http://vesale.info.fundp.ac.be/course/syllabus/7/32/Chap7, April 2001.

- [VRML2000] Ames, Andrea and Nadeau, David and Moreland, John. (1997) *VRML 2.0 Sourcebook* Second Edition

- [VRML97] *VRML 97*, International Specification ISO/IEC IS 14772-1, December 1997. http://www.vrml.org.

- [Ware2000] Ware, Colin. (2000) *Information Visualization: Perception for Design*. Morgan Kaufman.

- [Wessona2000] Wesson, Janet (2000). *Designing for Multimedia: New Challenges and Opportunities*, 2èmes journées MULTIMĚDIA, Namur, Belgium, 11-12 September 2000.

- [Wessonb2000] Wesson, Janet (2000). *Lecture 7: Object-oriented Analysis/Design*. Human Computer Interaction 4 (WRHH402).

- [Winograd1995] Winograd, T. (1995). *From programming environments to environments for designing*. Communications of the ACM 38(6): 65-74.

- [Wright1999] Wright, William. (1999) *Research Report: Information Animation Applications in the Capital Market*. Pg 83 [Card1999]

- [Zhang99] Dr. Ping Zhang, *Effective decision making with effective Human Information Interaction : A cognitive* perspective, Proceeding of the Fourth Asia-Pacific Decision Science Institute Conference, 1999.

APPENDIX A

Evaluation of the interface

I. Introduction

To achieve this evaluation, we are going to make use of the Cognitive Walkthrough Method. This method is described in [Vesale2001] and is focussed around a task. Its aim is to show clearly the usability problems that a user meets during the realisation of a given task. "Visualize some data in a chart" will be the task we are going to study.

We will analyse a concrete case of this task (consisting of specific actions). In order not to complicate the analysis, we will not consider the recovery procedures (by using the buttons *Next>* and *<<Delete)* that are standard.

To achieve an evaluation via the **Cognitive Walkthrough Method**, we must define a scenario. That is to say a description of the way by which a person would use the system to perform the task.

To build the scenario, we will use the following information :

- The interface that will be evaluated in utility and usability terms.
- The task to analyse, mentioned just above.
- The task analysis, fulfilled in the chapter *Task Analysis* that contains in particular "The work's context" and "The Utility and Usability criteria".
- The concrete task's structure, described underneath (task factorisation into goals and sub-goals).

II. Concrete Task's structure

To realise the concrete task's structure, one need to break down the task into goals and sub-goals, and describe all the actions that form the final sub-goal. An action consists of an operation executed by a user via an interaction mean (a keyboard, a mouse and so on). Furthermore, the action must have an impact on the interface (when it triggers a feedback), and/or on the system's state.

After having described the factorisation into goals/sub-goals, we will describe the sequence of actions that form each not decomposable sub-goal.

II.1 Factorisation into goals/sub-goals

1. Visualise some data in a Chart
   - 1.1 Choose a Chart's type
   - 1.2 Choose a title and some columns
     - 1.2.1 Choose a title
     - 1.2.2 Add the column **Gender**
     - 1.2.3 Add the column **Race**
     - 1.2.4 Add the column **Arithmetic_2000**
     - 1.2.5 Validate the choices
   - 1.3 Select the constraints for each column
     - 1.3.1 Select constraints for the column **Gender**
       - 1.3.1.1 Select **F**
     - 1.3.2 Select constraints for the column **Race**
       - 1.3.2.1 Select **Black**
       - 1.3.2.2 Select **White**
     - 1.3.3 Select constraints for the column **Arithmetic_2000**
       - 1.3.3.1 Select the >= comparison sign
       - 1.3.3.2 Fill in the field with **50**
     - 1.3.4 Validate the choices
   - 1.4 Associate each column with a dimension
     - 1.4.1 Associate the column **Gender** with **shape**
       - 1.4.1.1 Choose the dimension **shape**
       - 1.4.1.2 Customise the association
         - 1.4.1.2.1 Associate **F** with **Sphere**
           - 1.4.1.2.1.1 Select **F**
           - 1.4.1.2.1.2 Select ***Sphere***
           - 1.4.1.2.1.3 Association
         - 1.4.1.2.2 Validate the choices
     - 1.4.2 Associate the column **Race** with **color**
       - 1.4.2.1 Choose the dimension **color**
       - 1.4.2.2 Customise the association
         - 1.4.2.2.1 Associate **Black** with **Light Gray**
           - 1.4.2.2.1.1 Select **Black**
           - 1.4.2.2.1.2 Select **Light Gray**
           - 1.4.2.2.1.3 Association
         - 1.4.2.2.2 Associate **White** with **blue**
           - 1.4.2.2.2.1 Select **White**
           - 1.4.2.2.2.2 Select **blue**
           - 1.4.2.2.2.3 Association
         - 1.4.2.2.3 Validate the choices
     - 1.4.3 Associate the column **Arithmetic_2000** with **x-axis**
       - 1.4.3.1 Choose the dimension **x-axis**
     - 1.4.4 Validate the choices

II.2 Description of the final sub-goals

We do not describe all the final sub-goals because much of them are similar. But we describe sufficiently of them to be able to detect several problems.

Method to perform the sub-goal 1.2.1 : choose a title

Main but and active sub-goals :

1. Visualise some data in a Chart

1.2 Choose a title and some columns

1.2.1 : choose a title

Comments: /

	User's action	**Feedback**	**comments**
1.2.1.1	Erase the title.	• The title disappears.	**Screenshot one before the action**
1.2.1.2	Type a title.	• The title appears.	

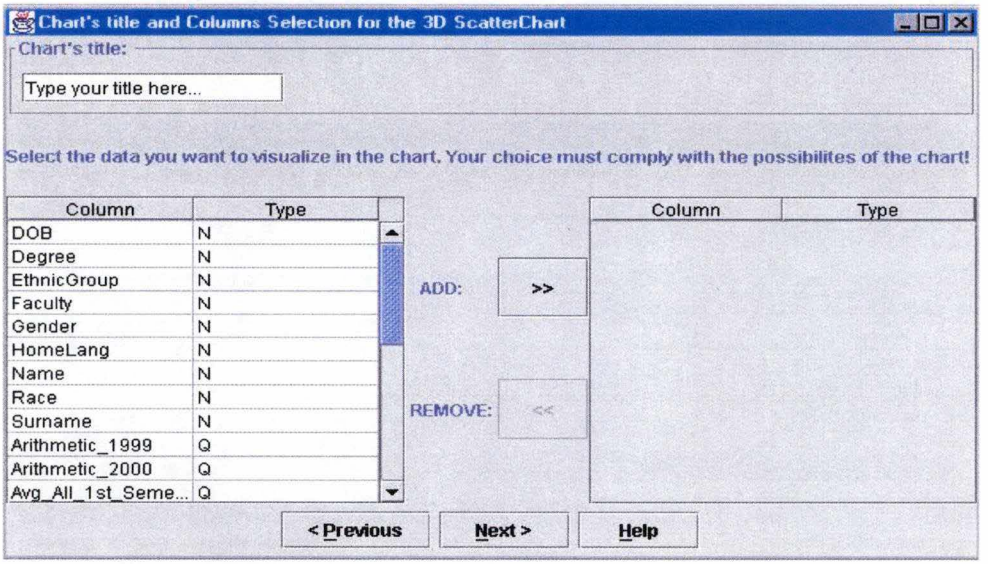

Figure 45: Screenshot one

Method to perform the sub-goal 1.2.2 : Add the column *Gender*

Main but and active sub-goals :

1. Visualise some data in a Chart

1.2 Choose a title and some columns

1.2.2 : Add the column *Gender*

Comments: /

	User's action	**Feedback**	**comments**
1.2.2.1	Move the mouse's cursor to the item **"Gender"** of the left list and click.	• This item is highlighted.	
1.2.2.2	Move the mouse's cursor to the button ">>" and click.	• The button is selected. • The item "Gender" disappears from the left list. • The item "Gender" appears in the right list. • The button "<<" becomes active.	**Screenshot two**

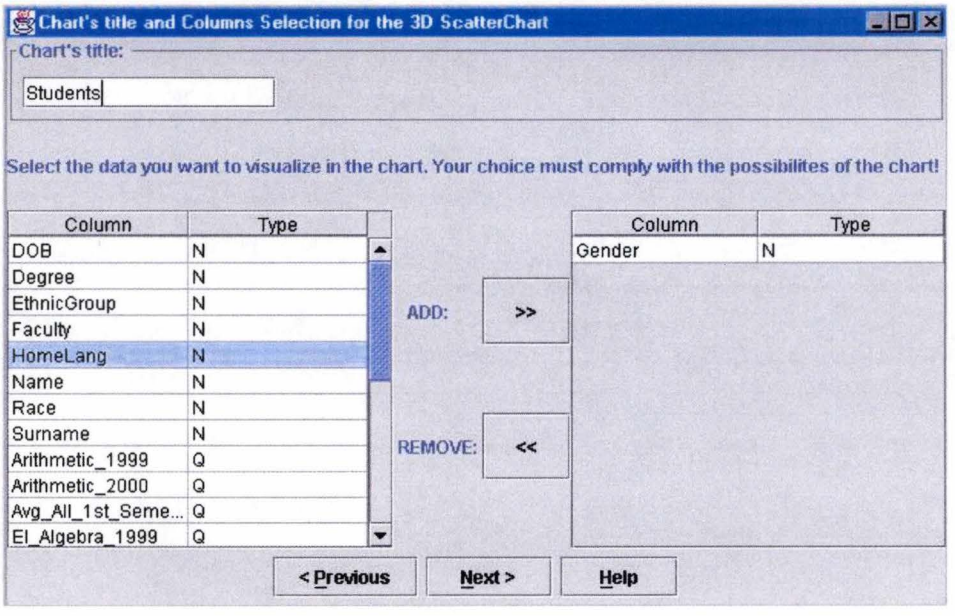

Figure 46: Screenshot two

Method to perform the sub-goal 1.2.5 : Validate the choices

Main but and active sub-goals :

1. Visualise some data in a Chart

1.2 Choose a title and some columns

1.2.5 : Validate the choices

Comments: /

	User's action	**Feedback**	**comments**
1.2.5.1	Move the mouse's cursor to the button "**Next**" and click.	• The button is selected. • The frame "**Constraints Selection**" appears.	

Method to perform the sub-goal 1.3.2.2 : Select *White*

Main but and active sub-goals :

1. Visualise some data in a Chart

1.3.2 Select constraints for the column *Race*

1.3.2.2 : Select White

Comments: /

	User's action	**Feedback**	**comments**
1.3.2.2.1	Move the mouse's cursor to the JComboBox "**Race**" and click.	• The JComboBox unrolls.	
1.3.2.2.2	Move the mouse's cursor to the item "**White**" and click.	• The item "**White**" is highlighted.	**Screenshot three**
1.3.2.2.3	Move the mouse's cursor to the button ">>" and click.	• The button is selected. • The item "**White**" disappears from the left JComboBox and "**Other**" appears. • "**White**" is added to the right JComboBox.	**Screenshot four** **Not visible**

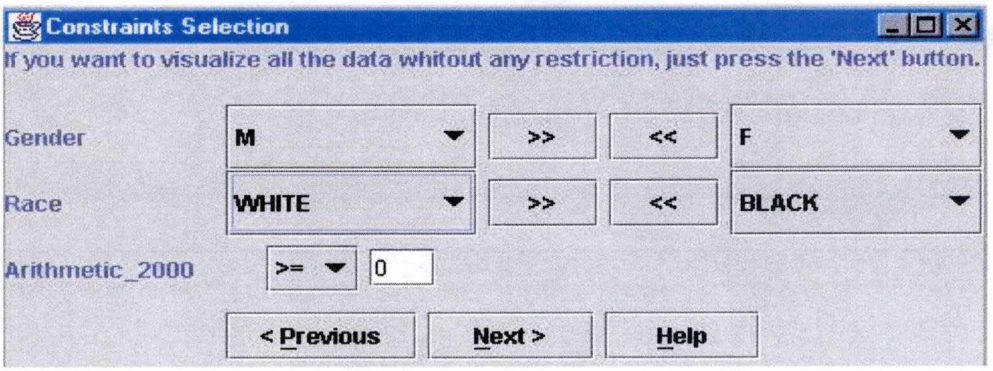

Figure 47: Screenshot three

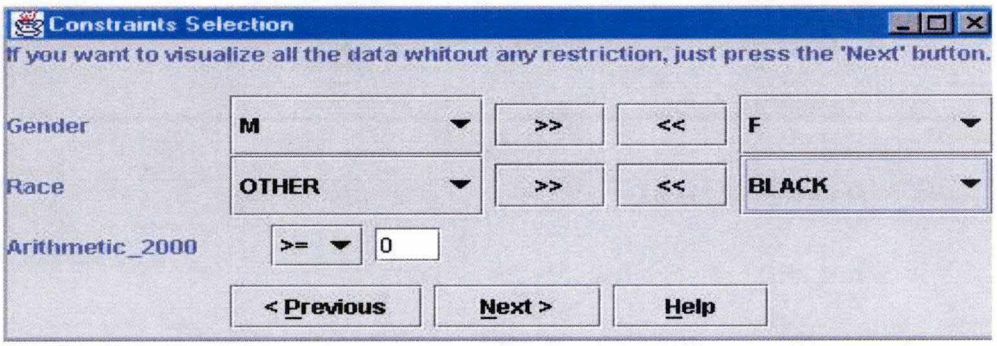

Figure 48: Screenshot four

Method to perform the sub-goal 1.4.1.1 : Choose the dimension *shape*

Main but and active sub-goals :

1. Visualise some data in a Chart

1.4.1 : Associate the column *Gender* with *shape*

1.4.1.1 : Choose the dimension *shape*

Comments: /

	User's action	**Feedback**	**comments**
1.4.1.1.1	Move the mouse's cursor to the JComboBox next to "**Gender**" and click.	• The JComboBox unrolls.	
1.4.1.1.2	Move the mouse's cursor to the item "**shape**" and click.	• The item "shape" is highlighted.	**Screenshot five**
1.4.1.1.3	Move the mouse's cursor to the button "**Associate**" and click.	• The button is selected. • The frame "**Customized Association**" appears.	**Screenshot six**

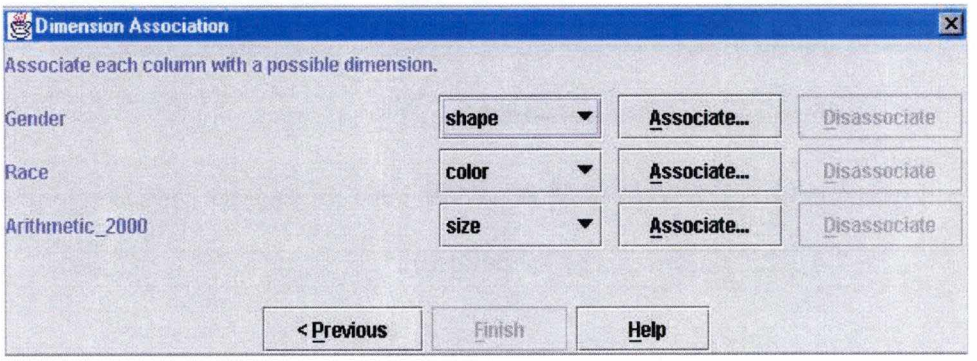

Figure 49: Screenshot five

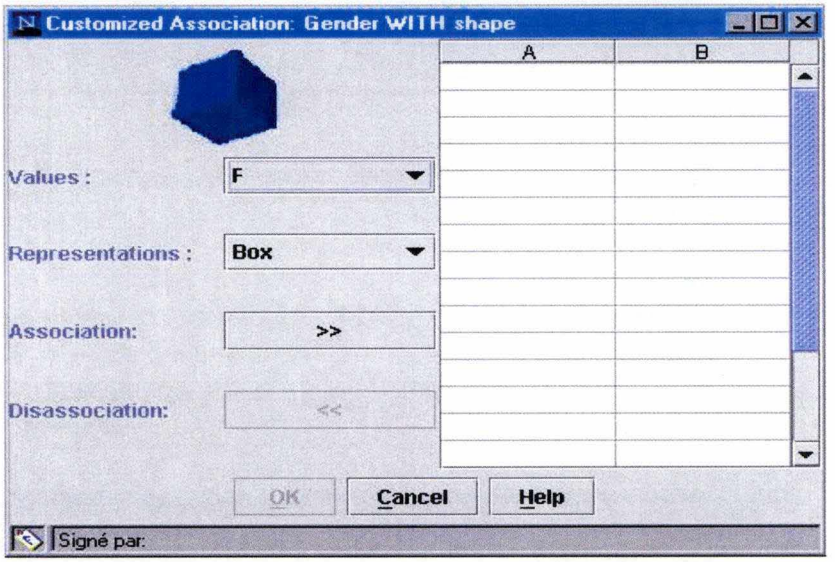

Figure 50: Screenshot six

Method to perform the sub-goal 1.4.1.2.1.2 : Select *Sphere*

Main but and active sub-goals :

1. Visualise some data in a Chart

1.4.1.2 Customise the association

1.4.1.2.1 : Associate *F* with *Sphere*

1.4.1.2.1.2 : Select *Sphere*

Comments: /

	User's action	**Feedback**	**comments**
1.4.1.2.1.2 .1	Move the mouse's cursor to the JComboBox "**Representation**" and click.	• The JComboBox unrolls.	
1.4.1.2.1.2 .2	Move the mouse's cursor to the item "**Sphere**" and click.	• The item "**Sphere**" is highlighted. • The displayed shape is replaced by a Sphere.	**Screenshot seven**

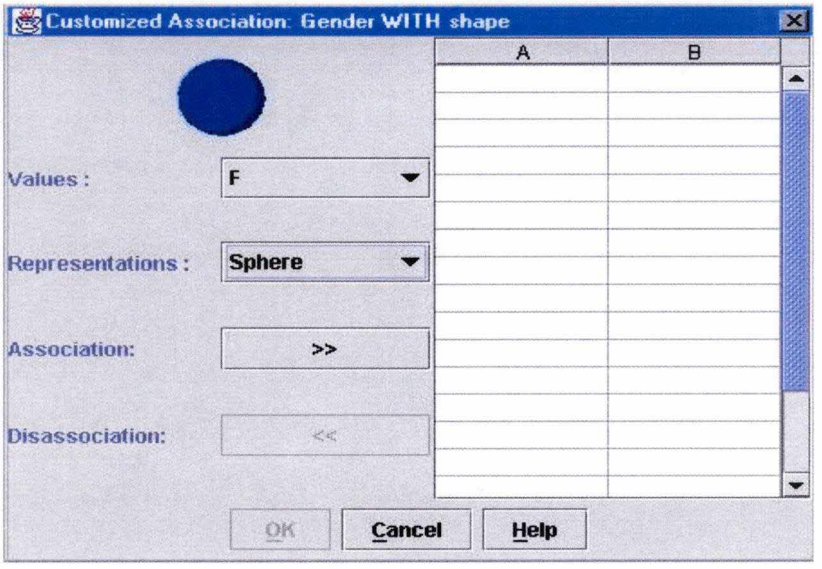

Figure 51: Screenshot seven

Method to perform the sub-goal 1.4.1.2.1.3 : Association

Main but and active sub-goals :

1. Visualise some data in a Chart

1.4.1.2 Customise the association

1.4.1.2.1 : Associate *F* with *Sphere*

1.4.1.2.1.3 : Association

Comments: /

	User's action	**Feedback**	**comments**
1.4.1.2.1.3.1	Move the mouse's cursor to the button ">>" and click.	• The button ">>" is selected. • The button ">>" becomes inactive. • The button "<<" becomes active. • The item "**F**" disappears from the JComboBox "**Values**". • The item "**Sphere**" disappears from the JComboBox "**Representation**". • A Cylinder is displayed. • The association appears in the list. • The button "**OK**" becomes active.	**Screenshot eight**

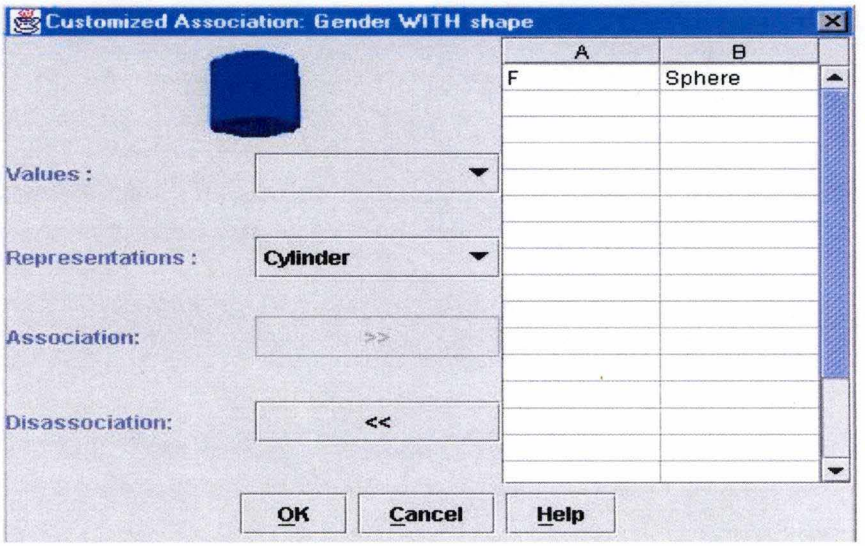

Figure 52: Screenshot eight

Method to perform the sub-goal 1.4.1.2.2: Validate the choices

Main but and active sub-goals :

1. **Visualise some data in a Chart**

1.4.1 Associate the column *Gender* with *shape*

1.4.1.2 Customize the association

1.4.1.2.2: Validate the choices

Comments: /

	User's action	**Feedback**	**comments**
1.4.1.2.2.1	Move the mouse's cursor to the button "**OK**" and click.	• The button is selected. • The current frame disappears. • The frame "**Dimension Association**" appears. • The item "**shape**" is removed from both JComboBoxes next to "**Race**" and "**Arithmetic_2000**". • The button "**Associate**" and the JComboBox next to "Gender" become inactive and the button "**Disassociate**" active.	**Screenshot nine** **Not visible**

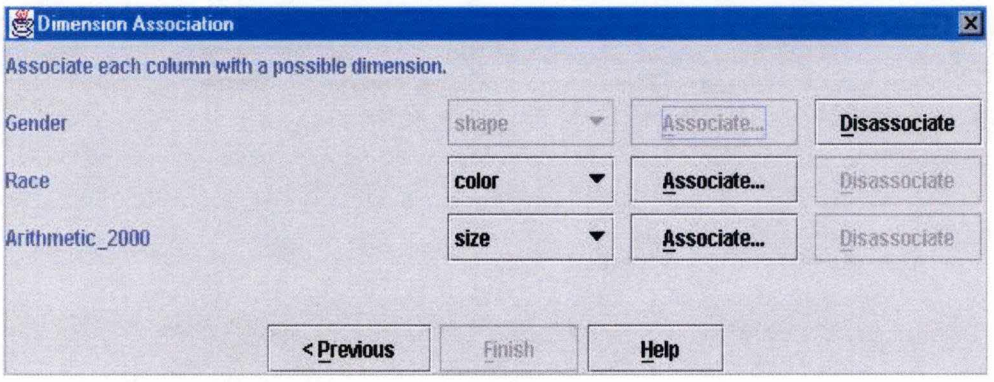

Figure 53: Screenshot nine

Method to perform the sub-goal 1.4.3.1 : Choose the dimension *x-axis*

Main but and active sub-goals :

1. Visualise some data in a Chart

1.4.3 : Associate the column *Arithmetic_2000* with *x-axis*

1.4.3.1 : Choose the dimension *x-axis*

Comments: /

	User's action	**Feedback**	**comments**
1.4.3.1.1	Move the mouse's cursor to the JComboBox next to "**Arithmetic_2000**" and click.	• The JComboBox unrolls.	
1.4.3.1.2	Move the mouse's cursor to the item "**x-axis**" and click.	• The item "**x-axis**" is highlighted.	
1.4.3.1.3	Move the mouse's cursor to the button "**Associate**" and click.	• The button is selected. • A message informs him that the association has been registered. • The item "**x-axis**" is removed from both JComboBoxes next to "**Gender**" and "**Race**".	**Not visible**
1.4.3.1.4	Move the mouse's cursor to the button "**OK**" and click.	• The button "**Associate**" and the JComboBox become inactive and the button "**Disassociate**" active. • The button "**Finish**" becomes active.	**Screenshot ten**

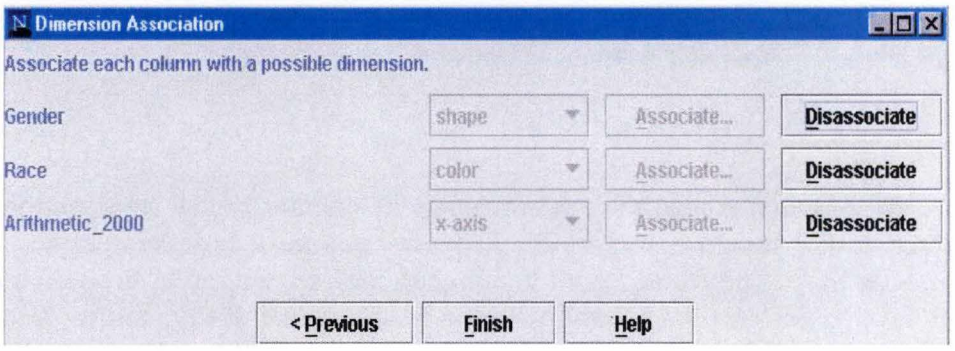

Figure 54: Screenshot ten

III. Questions raised by the method

The **Cognitive Walkthrough Method** consists of a series of questions to ask systematically for each element of the task's structure. Answering these questions enables to find out some problems of usability. They enable to estimate the distance between the physical universe and the psychological variables, around a specific task.

The first two questions of the method concern the task's structure into goals/sub-goals. They enable to identify the distance between the **abstract** task's structure, such as the user thinks of it, and the **concrete** task's structure, such as implemented by the system. The questions concerning the sub-goals are:

- *Is the user going to attempt to perform the adequate sub-goal?*
- *What is the needed knowledge to perform the adequate sub-goal? Will the user have this knowledge?*

After having answered these questions, for each final sub-goal, the analysis must go through the sequence of actions that form it.

The questions concerning the actions are:

- *Is the user going to perceive that the adequate action is available?*
- *Is the user going to associate this action with the sub-goal that he tries to perform?*
- *Is the user going to perceive the feedback?*
- *Is the user going to understand the feedback?*
- *Is the user going to see that he has progressed in the completion of the active sub-goals and the main goal?*

We have asked the questions for each sub-goals described in the concrete task's structure (goals/sub-goals), and for each action that forms the nine final sub-goals described above. We present here the most significant answers. The actions and sub-goals that are potentially sources of problems, are in **bold**.

III.1 Questions for the sub-goals

The user will be often guided to choose which is the next sub-goal (the next one in the concrete task's structure) to perform. This one is imposed by the interface. For instance, after having performed the 1.1 sub-goal, the user will go to the next frame and will have to perform the 1.2 sub-goal. So he cannot be wrong. In this case, he will always accomplish the adequate sub-goal.

- For the sub-goal 1.2.1 "*Choose a title*", the user will realise it because it is highlighted by a specific border that he cannot ignore, and because this field has the focus (see the screenshot one). Mind you that this sub-goal may be done after the 1.2.2 ... 1.2.4 sub-goals.

- The sub-goal 1.2.2 "*Add the column Gender*" follows logically the sub-goal 1.2.1. Furthermore, the column's selection from a list is a usual action for the user.

- 1.2.5 "*Validate the choices*" follows logically the sub-goals 1.2.1 ... 1.2.4. The user will have to use the standard button "*Next>*".

- 1.3.1 "*Select constraints for the column Gender*". **It is possible the user do not understand the goal of the frame in which he can enter his constraints.** However we have supposed (cf. the task's analysis) the user will often use this application. So, this problem will be overcome by trials and errors. Furthermore, he will quickly understand the aim of the frame if he knows his database's values. Besides, restrict the data set to visualize is a significant action for the user. The problem is so minimum. (see the screenshot three)

- 1.3.2.2 "*Select White*". This action follows logically "Select *Black*" The item's selection is an usual action for the user.

- 1.3.3.1 "Select >=" is a non common symbol for the user. Besides, it looks like the ">>" button. The string "is superior or equals to" would have been more suitable. **However, this problem is not insurmountable.**

- 1.3.3.2 "Fill in the field with 50". The user will be able to do it if he interprets correctly the symbol ">=" **(see above). Maybe, there won't be any problems because the user knows the database's columns (and their type).**

- **The actions 1.4.1 ... 1.4.3 "*Associate a column with a dimension*" can be performed in any order. The user has seen an example of a ScatterChart in the first window, so this sub-goal is not surprising. Furthermore, this type of association is common for him. In addition, the user has no choice: to go on his task, he will have to perform these associations (by pressing the buttons "*Associate*")**

- **1.4.3.1 "*Choose the dimension x-axis*". A button** Associate **instead of** Associate... **would have been better. But it does not raise a real problem.**

III.2 Questions for the actions

The user can easily perceive the adequate actions, and understand the feedback. The task's experience enables him to associate each action with the sub-goal that he tries to perform and to see that he has progressed in the completion of the active sub-goals.

There is maybe just an exception. The action 1.3.2.2.3 "Move the mouse's cursor to the button '>>' and click" can raise a problem. There are three feedback:

1) The button is selected.
2) The item "White" disappears from the left JComboBox
3) "White" is added to the right JComboBox.

The user can not see the third feedback without verifying (by clicking) the content of the JComboBox. If he does not see the second, he does not know the action has been realised. The item "White" should be selected in the JComboBox.

Due to the user's stereotype, we can assume there is no particular problem. He is used to perform this kind of actions and the feedback are clear enough to be correctly interpreted.

APPENDIX B

How To Launch the Visualization Application?

I. The VRML Browser

As a VRML browser, we have used the **Cosmo Player 2.1.1**© (1997-1999 **PLATINUM** technology, incorporated company). One can employ any other one that enables you to see and explore 3D worlds, but it has to provide an API of Java classes including classes related to the **External Authoring Interface**[46] (EAI) in order to allow the building of VRML objects from a Java code.

Cosmo Player is available from the *Cosmo Player Support Resources* page (http://cosmosoftware.com/support/player), on the *Cosmo Software* web site (http://cosmosoftware.com). Of course, you will find all the necessary information to install this browser in the **Release Notes**, provided with the player. You will also find help facilities that show you how to navigate through a virtual world, using their *navigation tool* (or *dashboard*).

Cosmo Player 2.1.1 is designed to work as a plug-in component for the following Web browsers:

- **Netscape 3.01 and higher**
- **Netscape 4.x**
- **Internet Explorer 4.x**

But, due to some compatibility problems with the User Interface, we have only used the second possibility (namely **Netscape Communicator 4.7**).

With **Cosmo Player**, you will obtain a zip file (**npcosmop211.zip**) that contains the EAI Java classes. We make extensive use of these classes in the implementation of the *Visualization Application*. Especially, in the chart classes, when we generate VRML objects, and add them to a virtual world. You have two possibilities to activate these classes:

- you can add these classes into the **rt.jar** file. We have used the Java Development Kit 1.2.1 for the application, and this file is located in the directory **\jdk1.2.1\jre\lib**.

- or you can add the path that leads to the zip file (in the **\Program Files\CosmoSoftware\CosmoPlayer** directory) to your classpath (see the **Release Notes** for more).

[46] The EAI is a scripting language integrated within *VRML*. It allows the programmer to control from a *Java* applet running in a web page (display by a Web browser) the content of a *VRML* browser window embedded on the same web page. The EAI is a proposed Informative Annex to the *VRML* specification [Darville&2000].

II. The Web Browser

As mentioned above, we have used the **Netscape Communicator 4.7** as the Web browser. It is in this browser that we will run the **EAI Applet**, that initialises the **Cosmo Player** and the virtual world.

The User Interface is mostly based on the **Swing Components**. But, as **Netscape 4.7** does not support by default these components, you need to add a file of *Java* classes (**Swing.jar** available on http://www.java.sun.com/products/) to the **Program Files\Netscape\Communicator\Program\Java\Classes** directory to enable swing.

And that's all for the Web browser.

III. The Database Component

To connect to the database, an *ODBC data source* must be registered with the system through the **ODBC Data Sources** option in the Windows **Control Panel**. *ODBC* is a technology developed by Microsoft to allow generic access to disparate database systems on the Windows platform (and some Unix platforms). The Java 2 Software Development Kit comes with a driver to allow any Java program to access any *ODBC* data source.

To run the current application, the database will have to be named **DataStudents**. The table contained in this database must be named **Results**, and when you register this database as an *ODBC data source*, you have to give the username "**anonymous**", and the password "**guest**". Doing this will allow the Java classes (of the application) to access the database, and to submit queries.

IV. The Java Classes

Now, you can put all the Java classes we have created in the same directory, and compile them with the **javac** utility. (**javac *.java** under MS-Dos).

Eventually, you must put the ***.class** files generated in the same directory as the ***.html**, the ***.wrl** and the ***.jpeg** files we have provided you. The html file (named Visualization Application) will be responsible to launch the **EAI Applet**, and to load the default VRML world (contained in the **root.wrl** file) into the Netscape browser. The ***.jpeg** files contain all the textures you may use in your charts, as a possible dimension.

You need to follow all these instructions only one time. Then, each time you want to launch the *Visualization Application*, you just need to open the ***.html** file: the default virtual world (three axes) will be opened via **Cosmo Player**, as well as the User Interface.

APPENDIX C

Main Java Classes

I. The Chart Class

```
import java.util.*;
import vrml.external.*;
import vrml.external.field.*;
import vrml.external.exception.*;
import java.awt.*;

public abstract class Chart {

        Vector semantics;// Each instance in this vector will describe the semantics of each
        selected column, related to a dimension of the data.

 String title;

 static Browser browser = null;// This is the browser that will display the VRML world
     containing the Chart. Browser is a VRML Object.

 static Node ancetre = null;// This object represents a basic VRML world to which we
     will add the chart. Node also is a VRML Object.

 EventInMFNode addtoanc = null;

 public Chart (String titlechart, Vector sem) {  //This constructor will initialise
     (mostly the legend on the axes) the VRML world.

    title = titlechart;
    semantics = sem;

    // initialization of the vrml world that will contain the chart

    // adding the title

    Node[] transftitle = browser.createVrmlFromString("Transform{}");
    Node[] shapetitle  = browser.createVrmlFromString("Shape{}");
    Node[] titleno = browser.createVrmlFromString("Text{fontStyle FontStyle"+
                     "{justify \"END\" family \"COMIC " +
                     "SANS MS\" style \"BOLD\" size 1.0 }}");

    EventInSFVec3f set_postitle = (EventInSFVec3f)
                     transftitle[0].getEventIn("translation");
    EventInMFString set_title = (EventInMFString) titleno[0].getEventIn("string");
    float ptit[] = new float[3];
    ptit[0] = 0.0f;
    ptit[1] = 13.0f;
```

```
ptit[2] = 0.0f;
set_postitle.setValue(ptit);
set_title.set1Value(0, title);

addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
EventInMFNode addtotransftitle = (EventInMFNode)
                transftitle[0].getEventIn("addChildren");
EventInSFNode addtoshape_title = (EventInSFNode)
                shapetitle[0].getEventIn("geometry");
addtoshape_title.setValue(titleno[0]);
addtotransftitle.set1Value(0,shapetitle[0]);
addtoanc.set1Value(1,transftitle[0]);

// now we must go through each element of the sem vector to create
// everything we want to see on the chart!

semantics.trimToSize();
Enumeration enum = semantics.elements();
int i = 0;

while ( enum.hasMoreElements() )
{
  Semantic s = (Semantic) enum.nextElement();
  String dim = s.get_rep().get_dimension();

  if (dim.equalsIgnoreCase("x-axis"))
  {
     // The x-axis' legend

     addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
     Node[] transflegxaxisaxe = browser.createVrmlFromString("Transform{}");
     Node[] shapelegxaxisaxe  = browser.createVrmlFromString("Shape{}");
     Node[] legxaxisaxe = browser.createVrmlFromString("Text{fontStyle
                FontStyle" + "{justify \"BEGIN\" family \"TYPEWRITER\" "
                +"style \"BOLD ITALIC\" size 0.5 }}");
     EventInSFVec3f set_poslegxaxisaxe = (EventInSFVec3f)
                transflegxaxisaxe[0].getEventIn("translation");
     EventInMFString set_legxaxisaxe = (EventInMFString)
                legxaxisaxe[0].getEventIn("string");
     float postitaxe[] = new float[3];
      Float poslegx = new Float (s.get_rep().get_dimvalues_at(
                (s.get_rep().datavalues.length)-1));
     postitaxe[0] = poslegx.floatValue() + 2.0f;
     postitaxe[1] = -0.5f ;
     postitaxe[2] = 0.0f;
     set_poslegxaxisaxe.setValue(postitaxe);

     set_legxaxisaxe.set1Value(0,s.get_name());
     EventInMFNode addtotransflegxaxisaxe = (EventInMFNode)
                transflegxaxisaxe[0].getEventIn("addChildren");
```

```
EventInSFNode addtoshape_legxaxisaxe = (EventInSFNode)
            shapelegxaxisaxe[0].getEventIn("geometry");
addtoshape_legxaxisaxe.setValue(legxaxisaxe[0]);
addtotransflegxaxisaxe.set1Value(0,shapelegxaxisaxe[0]);
addtoanc.set1Value(1,transflegxaxisaxe[0]);

// axis' graduation

if ( (s.get_type().equalsIgnoreCase("nominal")) ||
   (s.get_type().equalsIgnoreCase("ordinal")) )
{
 for (int k=0; k < s.get_rep().datavalues.length; k++)
 {
        Node[] transflegaxis = browser.createVrmlFromString("Transform{}");
        Node[] shapelegaxis  = browser.createVrmlFromString("Shape{}");
        Node[] legaxis = browser.createVrmlFromString("Text{fontStyle
              FontStyle"+ "{justify \"MIDDLE\" family \"SANS\" " +
             "size 0.5 }}");
        EventInSFVec3f set_poslegaxis = (EventInSFVec3f)
              transflegaxis[0].getEventIn("translation");
        EventInMFString set_legaxis = (EventInMFString)
              legaxis[0].getEventIn("string");
        float postitxaxis[] = new float[3];
        Float t = new Float(s.get_rep().get_dimvalues_at(k));
        postitxaxis[0] = t.floatValue();
        postitxaxis[1] = -1.0f;
        postitxaxis[2] = 0.0f;
        set_poslegaxis.setValue(postitxaxis);
        String axismark = (s.get_rep().get_datavalues_at(k).length() > 8 ?
              s.get_rep().get_datavalues_at(k).substring(0, 7) :
              s.get_rep().get_datavalues_at(k));
        set_legaxis.set1Value(0,axismark);
        addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
        EventInMFNode addtotransflegaxis = (EventInMFNode)
              transflegaxis[0].getEventIn("addChildren");
        EventInSFNode addtoshape_legaxis = (EventInSFNode)
              shapelegaxis[0].getEventIn("geometry");
        addtoshape_legaxis.setValue(legaxis[0]);
        addtotransflegaxis.set1Value(0,shapelegaxis[0]);
        addtoanc.set1Value(1,transflegaxis[0]);
 }
}
else
{
 for (int k=0; k < s.get_rep().datavalues.length; k++)
 {
        Node[] transflegaxis = browser.createVrmlFromString("Transform{}");
        Node[] shapelegaxis  = browser.createVrmlFromString("Shape{}");
        Node[] legaxis = browser.createVrmlFromString("Text{fontStyle
              FontStyle" +  "{justify \"MIDDLE\" family \"SANS\" " +
```

```
                "size 0.5 }}");
            EventInSFVec3f set_poslegaxis = (EventInSFVec3f)
                transflegaxis[0].getEventIn("translation");
            EventInMFString set_legaxis = (EventInMFString)
                legaxis[0].getEventIn("string");
            float postitxaxis[] = new float[3];
            Float t = new Float (s.get_rep().get_dimvalues_at(k));
            postitxaxis[0] = t.floatValue();
            postitxaxis[1] = -1.0f;
            postitxaxis[2] = 0.0f;
            set_poslegaxis.setValue(postitxaxis);

            String axismark = (s.get_rep().get_datavalues_at(k).length() > 8 ?
                s.get_rep().get_datavalues_at(k).substring(0, 7) :
                s.get_rep().get_datavalues_at(k));
            set_legaxis.set1Value(0,axismark);
            addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
            EventInMFNode addtotransflegaxis = (EventInMFNode)
                transflegaxis[0].getEventIn("addChildren");
            EventInSFNode addtoshape_legaxis = (EventInSFNode)
                shapelegaxis[0].getEventIn("geometry");
            addtoshape_legaxis.setValue(legaxis[0]);
            addtotransflegaxis.set1Value(0,shapelegaxis[0]);
            addtoanc.set1Value(1,transflegaxis[0]);
        }
    }
};//end of "x-axis"

if (dim.equalsIgnoreCase("y-axis"))
{
    // The y-axis's legend

    addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
    Node[] transflegyaxisaxe = browser.createVrmlFromString("Transform{}");
    Node[] shapelegyaxisaxe  = browser.createVrmlFromString("Shape{}");
    Node[] legyaxisaxe = browser.createVrmlFromString("Text{fontStyle
            FontStyle" + "{justify \"END\" family \"TYPEWRITER\" " +
            "style \"BOLD ITALIC\"  size 0.5 }}");
    EventInSFVec3f set_poslegyaxisaxe = (EventInSFVec3f)
            transflegyaxisaxe[0].getEventIn("translation");
    EventInMFString set_legyaxisaxe = (EventInMFString)
            legyaxisaxe[0].getEventIn("string");
    float postitaxe[] = new float[3];
    Float poslegy = new Float(s.get_rep().get_dimvalues_at(
            (s.get_rep().datavalues.length)-1));
    postitaxe[0] = -0.5f;
    postitaxe[1] = poslegy.floatValue() + 2.0f;
    postitaxe[2] = 0.0f;
    set_poslegyaxisaxe.setValue(postitaxe);
```

```
set_legyaxisaxe.set1Value(0,s.get_name());
EventInMFNode addtotransflegyaxisaxe = (EventInMFNode)
        transflegyaxisaxe[0].getEventIn("addChildren");
EventInSFNode addtoshape_legyaxisaxe = (EventInSFNode)
        shapelegyaxisaxe[0].getEventIn("geometry");
addtoshape_legyaxisaxe.setValue(legyaxisaxe[0]);
addtotransflegyaxisaxe.set1Value(0,shapelegyaxisaxe[0]);
addtoanc.set1Value(1,transflegyaxisaxe[0]);

// axis' graduation

if ( (s.get_type().equalsIgnoreCase("nominal")) ||
   (s.get_type().equalsIgnoreCase("ordinal")) )
{
 for (int k=0; k < s.get_rep().datavalues.length; k++)
 {
        Node[] transflegaxis = browser.createVrmlFromString("Transform{}");
        Node[] shapelegaxis  = browser.createVrmlFromString("Shape{}");
        Node[] legaxis = browser.createVrmlFromString("Text{fontStyle
                FontStyle" + "{justify \"MIDDLE\" family \"SANS\" " +
                "size 0.5 }}");
        EventInSFVec3f set_poslegaxis = (EventInSFVec3f)
                transflegaxis[0].getEventIn("translation");
        EventInMFString set_legaxis = (EventInMFString)
                legaxis[0].getEventIn("string");
        float postityaxis[] = new float[3];
        postityaxis[0] = -1.0f;
        Float t = new Float (s.get_rep().get_dimvalues_at(k));
        postityaxis[1] = t.floatValue();
        postityaxis[2] = 0.0f;
        set_poslegaxis.setValue(postityaxis);

        String axismark = (s.get_rep().get_datavalues_at(k).length() > 8 ?
                s.get_rep().get_datavalues_at(k).substring(0, 7) :
                s.get_rep().get_datavalues_at(k));
        set_legaxis.set1Value(0,axismark);

        addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
        EventInMFNode addtotransflegaxis = (EventInMFNode)
                transflegaxis[0].getEventIn("addChildren");
        EventInSFNode addtoshape_legaxis = (EventInSFNode)
                shapelegaxis[0].getEventIn("geometry");
        addtoshape_legaxis.setValue(legaxis[0]);
        addtotransflegaxis.set1Value(0,shapelegaxis[0]);
        addtoanc.set1Value(1,transflegaxis[0]);
  }
}
else
{
 for (int k=0; k < s.get_rep().datavalues.length; k++)
```

```
    {
        Node[] transflegaxis = browser.createVrmlFromString("Transform{}");
        Node[] shapelegaxis  = browser.createVrmlFromString("Shape{}");
        Node[] legaxis = browser.createVrmlFromString("Text{fontStyle
            FontStyle" + "{justify \"MIDDLE\" family \"SANS\" " +
            "size 0.5 }}");

        EventInSFVec3f set_poslegaxis = (EventInSFVec3f)
            transflegaxis[0].getEventIn("translation");
        EventInMFString set_legaxis = (EventInMFString)
            legaxis[0].getEventIn("string");

        float postityaxis[] = new float[3];
        Float t = new Float (s.get_rep().get_dimvalues_at(k));
        postityaxis[1] = t.floatValue();
        postityaxis[0] = -1.0f;
        postityaxis[2] = 0.0f;
        set_poslegaxis.setValue(postityaxis);

        String axismark = (s.get_rep().get_datavalues_at(k).length() > 8 ?
            s.get_rep().get_datavalues_at(k).substring(0, 7) :
            s.get_rep().get_datavalues_at(k));
        set_legaxis.set1Value(0,axismark);

        addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
        EventInMFNode addtotransflegaxis = (EventInMFNode)
            transflegaxis[0].getEventIn("addChildren");
        EventInSFNode addtoshape_legaxis = (EventInSFNode)
            shapelegaxis[0].getEventIn("geometry");
        addtoshape_legaxis.setValue(legaxis[0]);
        addtotransflegaxis.set1Value(0,shapelegaxis[0]);
        addtoanc.set1Value(1,transflegaxis[0]);
    }
  }
}; //end of "y-axis"

if (dim.equalsIgnoreCase("z-axis"))
{
  // The axe's legend

  addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
  Node[] transflegzaxisaxe = browser.createVrmlFromString("Transform{}");
  Node[] shapelegzaxisaxe  = browser.createVrmlFromString("Shape{}");
  Node[] legzaxisaxe = browser.createVrmlFromString("Text{fontStyle
        FontStyle" + "{justify \"END\" family \"TYPEWRITER\" " +
       "style \"BOLD ITALIC\" size 0.5 }}");
  EventInSFVec3f set_poslegzaxisaxe = (EventInSFVec3f)
        transflegzaxisaxe[0].getEventIn("translation");
  EventInSFRotation set_rotlegzaxisaxe = (EventInSFRotation)
        transflegzaxisaxe[0].getEventIn("rotation");
```

```
EventInMFString set_legzaxisaxe = (EventInMFString)
        legzaxisaxe[0].getEventIn("string");
float postitaxe[] = new float[3];
Float poslegz = new Float(s.get_rep().get_dimvalues_at(
        (s.get_rep().datavalues.length)-1));
postitaxe[0] = 0.0f;
postitaxe[1] = -0.5f;
postitaxe[2] = poslegz.floatValue() + 0.5f;
set_poslegzaxisaxe.setValue(postitaxe);
float rotation[] = {0.0f, 1.0f, 0.0f, 1.6f};
set_rotlegzaxisaxe.setValue(rotation);

set_legzaxisaxe.set1Value(0,s.get_name());
EventInMFNode addtotransflegzaxisaxe = (EventInMFNode)
        transflegzaxisaxe[0].getEventIn("addChildren");
EventInSFNode addtoshape_legzaxisaxe = (EventInSFNode)
        shapelegzaxisaxe[0].getEventIn("geometry");
addtoshape_legzaxisaxe.setValue(legzaxisaxe[0]);
addtotransflegzaxisaxe.set1Value(0,shapelegzaxisaxe[0]);
addtoanc.set1Value(1,transflegzaxisaxe[0]);


if ( (s.get_type().equalsIgnoreCase("nominal")) ||
    (s.get_type().equalsIgnoreCase("ordinal")) )
{
  for (int k=0; k < s.get_rep().datavalues.length; k++)
  {
        Node[] transflegaxis = browser.createVrmlFromString("Transform{}");
        Node[] shapelegaxis  = browser.createVrmlFromString("Shape{}");
        Node[] legaxis = browser.createVrmlFromString("Text{fontStyle
                FontStyle" + "{justify \"MIDDLE\" family \"SANS\" " +
                "size 0.5 }}");
        EventInSFVec3f set_poslegaxis = (EventInSFVec3f)
                transflegaxis[0].getEventIn("translation");
        EventInSFRotation set_rotlegaxis = (EventInSFRotation)
                transflegaxis[0].getEventIn("rotation");
        EventInMFString set_legaxis = (EventInMFString)
                legaxis[0].getEventIn("string");
        float postitzaxis[] = new float[3];
        postitzaxis[0] = 0.0f;
        Float t = new Float (s.get_rep().get_dimvalues_at(k));
        postitzaxis[1] = -1.5f;
        postitzaxis[2] = t.floatValue();
        set_poslegaxis.setValue(postitzaxis);
        set_rotlegaxis.setValue(rotation);

        String axismark = (s.get_rep().get_datavalues_at(k).length() > 8 ?
                s.get_rep().get_datavalues_at(k).substring(0, 7) :
                s.get_rep().get_datavalues_at(k));
        set_legaxis.set1Value(0,axismark);
```

```
        addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
        EventInMFNode addtotransflegaxis = (EventInMFNode)
                transflegaxis[0].getEventIn("addChildren");
        EventInSFNode addtoshape_legaxis = (EventInSFNode)
                shapelegaxis[0].getEventIn("geometry");
        addtoshape_legaxis.setValue(legaxis[0]);
        addtotransflegaxis.set1Value(0,shapelegaxis[0]);
        addtoanc.set1Value(1,transflegaxis[0]);
  }
}
else
{
  for (int k=0; k < s.get_rep().datavalues.length; k++)
  {
        Node[] transflegaxis = browser.createVrmlFromString("Transform{}");
        Node[] shapelegaxis  = browser.createVrmlFromString("Shape{}");
        Node[] legaxis = browser.createVrmlFromString("Text{fontStyle
                FontStyle" + "{justify \"MIDDLE\" family \"SANS\" " +
                "size 0.5 }}");
        EventInSFVec3f set_poslegaxis = (EventInSFVec3f)
                transflegaxis[0].getEventIn("translation");
        EventInSFRotation set_rotlegaxis = (EventInSFRotation)
                transflegaxis[0].getEventIn("rotation");
        EventInMFString set_legaxis = (EventInMFString)
                legaxis[0].getEventIn("string");

        float postitzaxis[] = new float[3];
        Float t = new Float (s.get_rep().get_dimvalues_at(k));
        postitzaxis[0] = 0.0f;
        postitzaxis[1] = -1.5f;
        postitzaxis[2] = t.floatValue();
        set_poslegaxis.setValue(postitzaxis);
        set_rotlegaxis.setValue(rotation);

        String axismark = (s.get_rep().get_datavalues_at(k).length() > 8 ?
                s.get_rep().get_datavalues_at(k).substring(0, 7) :
                s.get_rep().get_datavalues_at(k));
        set_legaxis.set1Value(0,axismark);

        addtoanc = (EventInMFNode) ancetre.getEventIn("addChildren");
        EventInMFNode addtotransflegaxis = (EventInMFNode)
                transflegaxis[0].getEventIn("addChildren");
        EventInSFNode addtoshape_legaxis = (EventInSFNode)
                shapelegaxis[0].getEventIn("geometry");
        addtoshape_legaxis.setValue(legaxis[0]);
        addtotransflegaxis.set1Value(0,shapelegaxis[0]);
        addtoanc.set1Value(1,transflegaxis[0]);
  }
}
```

```
      }//end of "z-axis"
    } //end of while loop

    // Now it only remains to add all the objects to complete the chart
  }

  // this abstract method will have to be implemented by the concrete chart classes.
  It will manage the dynamic building of the virtual chart.

  public abstract void addRow(Datum data);

  public EventInMFNode getAddToAnc() {
   return addtoanc;
  }

  public Node getAncetre() {
   return ancetre;
  }

  public Browser getBrowser() {
   return browser;
  }

  public Vector getSemantics() {
   return semantics;
  }

  public String getTitle() {
   return title;
  }

  public static void set_browser (Browser b)
  {
    browser = b;
  }

  public static void set_ancetre (Node n)
  {
    ancetre = n;
  }

} //end of the abstract Chart class.
```

II. The ScatterChart Class

```
import java.util.*;
import vrml.external.*;
import vrml.external.field.*;
import vrml.external.exception.*;
import java.awt.*;

public class ScatterChart extends Chart implements EventOutObserver  {


public ScatterChart (String titlechart, Vector sem) { //calls the constructor of the
        Chart superclass. This will initialise (mostly the legend of the axes) the VRML
        world.

        super (titlechart, sem);
         //sem = Vector of Semantic Objects (containing the semantic of each
                column).
}

//The abstract method of the Chart class is now implemented by this class
public void addRow(Datum data)
  {
        /* The data parameter represents the current row to add to the chart.
        So we have to analyse the vector of data with regard to the semantics
        vector, and construct the object corresponding to this data into the vrml
        world. */

    // What do we need to draw an object in a virtual world?

     // the 3d-position
    float x = 0.0f, y = 0.0f, z = 0.0f;
    String xvalue = "", yvalue = "", zvalue = "";
    String xcol = "", ycol = "", zcol = "";

    //  the color
    float colors[]= {0.0f,0.0f,1.0f}; // blue by default
    String colorcol = "";
    String colorvalue = "";

    // The shape
    String theshape = "sphere";
    String shapecol = "";
    String shapevalue = "";

    // The texture
    String texture = "";
    String textcol = "";
    String textvalue = "";
```

```
// The size
float size = 0.3f;
String sizecol = "";
String sizevalue = "";

// The transparency
float transparency = 0.0f;
String transcol = "";
String transvalue = "";

Enumeration val = data.get_values().elements();
Enumeration se = semantics.elements();

//For each element of the row (of data), associate a Representation
while (val.hasMoreElements())
{
  String elem = val.nextElement().toString();
  Semantic colsem = (Semantic) se.nextElement();

  if (colsem.get_rep().get_dimension().equalsIgnoreCase("x-axis"))
  {
    xvalue = elem;
    xcol = colsem.get_name();
    if (colsem.get_type().equalsIgnoreCase("quantitative"))
    {
      Float pos = new Float(elem);
      Float a = new Float ((colsem.get_rep().datavalues[1]));
      Float b = new Float ((colsem.get_rep().dimvalues[1]));

      x = (pos.floatValue()) / (a.floatValue()/ b.floatValue());

    }
    else
    {
      int index = 0;
      for (int i=0; i < colsem.get_rep().datavalues.length; i++)
      {
        if (elem.equalsIgnoreCase(colsem.get_rep().datavalues[i]))
        {
          index = i;
        }
      };

      Float u = new Float (colsem.get_rep().dimvalues[index]);
      x = u.floatValue();
    }
  };
```

```
if (colsem.get_rep().get_dimension().equalsIgnoreCase("y-axis"))
{
 yvalue = elem;
 ycol = colsem.get_name();
 if (colsem.get_type().equalsIgnoreCase("quantitative"))
 {
  Float pos = new Float(elem);
  Float a = new Float ((colsem.get_rep().datavalues[1]));
  Float b = new Float ((colsem.get_rep().dimvalues[1]));
  y = (pos.floatValue()) / (a.floatValue()/ b.floatValue());
 }
 else
 {
  int index = 0;
  for (int i=0; i < colsem.get_rep().datavalues.length; i++)
  {
   if (elem.equalsIgnoreCase(colsem.get_rep().datavalues[i]))
   {
    index = i;
   }
  };

  Float u = new Float (colsem.get_rep().dimvalues[index]);
  y = u.floatValue();
 }
};

if (colsem.get_rep().get_dimension().equalsIgnoreCase("z-axis"))
 {
 zvalue = elem;
 zcol = colsem.get_name();
 if (colsem.get_type().equalsIgnoreCase("quantitative"))
 {
  Float pos = new Float(elem);
  Float a = new Float ((colsem.get_rep().datavalues[1]));
  Float b = new Float ((colsem.get_rep().dimvalues[1]));
  z = (pos.floatValue()) / (a.floatValue()/ b.floatValue());
 }
 else
 {
  int index = 0;
  for (int i=0; i < colsem.get_rep().datavalues.length; i++)
  {
   if (elem.equalsIgnoreCase(colsem.get_rep().datavalues[i]))
   {
    index = i;
   }
  };

  Float u = new Float (colsem.get_rep().dimvalues[index]);
```

```
    z = u.floatValue();
  }

};

if (colsem.get_rep().get_dimension().equalsIgnoreCase("color"))
{
  int index = 0;
  colorcol = colsem.get_name();
  colorvalue = elem;
  for (int i=0; i < colsem.get_rep().datavalues.length; i++)
  {
    if (elem.equalsIgnoreCase(colsem.get_rep().datavalues[i]))
    {
      index = i;
    }
  };
  Color color = Color.blue;
  color = getColor (colsem.get_rep().dimvalues[index].toLowerCase());

  colors[0] =  ((float) color.getRed()) / 255.0f;
  colors[1] =  ((float) color.getGreen()) / 255.0f;
  colors[2] =  ((float) color.getBlue()) / 255.0f;
};

if (colsem.get_rep().get_dimension().equalsIgnoreCase("shape"))
{
  int index = 0;
  shapecol = colsem.get_name();
  shapevalue = elem;
  for (int i=0; i < colsem.get_rep().datavalues.length; i++)
  {
    if (elem.equalsIgnoreCase(colsem.get_rep().datavalues[i]))
    {
      index = i;
    }
  };

    theshape = (colsem.get_rep().dimvalues[index]);
};

if (colsem.get_rep().get_dimension().equalsIgnoreCase("texture"))
{
  int index = 0;
  textcol = colsem.get_name();
  textvalue = elem;
  for (int i=0; i < colsem.get_rep().datavalues.length; i++)
  {
    if (elem.equalsIgnoreCase(colsem.get_rep().datavalues[i]))
    {
```

```
      index = i;
    }
   };

    texture = (colsem.get_rep().dimvalues[index]);
  };

  if (colsem.get_rep().get_dimension().equalsIgnoreCase("transparency"))
  {
   transcol = colsem.get_name();
   transvalue = elem;
   int index = 0;
   for (int i=0; i < colsem.get_rep().datavalues.length; i++)
   {
    if (elem.equalsIgnoreCase(colsem.get_rep().datavalues[i]))
    {
     index = i;
    }
   };

   Float u = new Float (colsem.get_rep().dimvalues[index]);
   transparency = u.floatValue();
  };

  if (colsem.get_rep().get_dimension().equalsIgnoreCase("size"))
  {
     sizecol = colsem.get_name();
     sizevalue = elem;
     Float temp = new Float (elem);

     Float a = new Float ((colsem.get_rep().datavalues[1]));
     Float b = new Float ((colsem.get_rep().dimvalues[1]));

     size = (temp.floatValue()) / (a.floatValue()/ b.floatValue());
  }
}; //end of while (val.hasMoreElements())

String geoShape = "";
if ( theshape.equalsIgnoreCase("sphere"))
{
 geoShape = "Sphere{ radius "+ size +"}";
};

if ( theshape.equalsIgnoreCase("box"))
{
 geoShape = "Box{ size "+ size + " " + size + " " + size +"}";
};

if ( theshape.equalsIgnoreCase("cylinder"))
{
```

```
  geoShape = "Cylinder{ radius "+ size + "height " + size +"}";
};

if ( theshape.equalsIgnoreCase("cone"))
{
  geoShape = "Cone{ bottomRadius "+ size + "height " + size +"}";
};

//get the browser that will manage the display of VRML objects
Browser browser = super.getBrowser();
//get the node to which the browser will add the new objects
Node ancetre = super.getAncetre();


 // 1. Getting the needed nodes
      //The main node of the 3D shape
           Node[] anch = browser.createVrmlFromString("Anchor{}");
           Node[] transf = browser.createVrmlFromString("Transform{}");

      // The nodes for the shape
           Node[] shape  = browser.createVrmlFromString("Shape{}");
           Node[] appear = browser.createVrmlFromString("Appearance{}");
           Node[] material = browser.createVrmlFromString("Material{}");
           Node[] text = browser.createVrmlFromString("ImageTexture{}");
           Node[] geom = browser.createVrmlFromString(geoShape);


// 2. Getting the attributes events
       // Attributes for the shape
           EventInSFColor set_color = (EventInSFColor)
                    material[0].getEventIn("diffuseColor");
           EventInSFFloat set_transp = (EventInSFFloat)
                    material[0].getEventIn("transparency");
           EventInSFVec3f set_pos = (EventInSFVec3f)
                    transf[0].getEventIn("translation");
           EventInMFString set_text = (EventInMFString)
                    text[0].getEventIn("url");
           EventInMFString set_anchor = (EventInMFString)
                    anch[0].getEventIn("url");
           EventInSFString set_descr = (EventInSFString)
                    anch[0].getEventIn("description");

 // 3. Giving the attributes values
           set_color.setValue(colors);
           set_transp.setValue(transparency);
           float position[] = {x, y, z};
           set_pos.setValue(position);
           String urlText[] = {texture};
           set_text.setValue(urlText);
```

```
                String urlAnchor[] = {""};
                set_anchor.setValue(urlAnchor);

                String description = buildDescription (xcol, xvalue, ycol, yvalue,
                        zcol, zvalue, sizecol, sizevalue, transcol, transvalue, colorcol,
                        colorvalue, shapecol, shapevalue, textcol, textvalue);
                set_descr.setValue(description);

        // 4. Getting the structure events
          // The main nodes of the whole scene
                EventInMFNode addtoancetre = (EventInMFNode)
                        ancetre.getEventIn("addChildren");
                EventInMFNode addtoanchor = (EventInMFNode)
                        anch[0].getEventIn("addChildren");

          // the nodes for the shape
                EventInMFNode addtotransf = (EventInMFNode)
                        transf[0].getEventIn("addChildren");
                EventInSFNode addtoshape_app = (EventInSFNode)
                        shape[0].getEventIn("appearance");
                EventInSFNode addtoshape_geom = (EventInSFNode)
                        shape[0].getEventIn("geometry");
                EventInSFNode addtoapp = (EventInSFNode)
                        appear[0].getEventIn("material");
                EventInSFNode addtexturetoapp = (EventInSFNode)
                        appear[0].getEventIn("texture");


        // 5. Building the structure
                addtoapp.setValue(material[0]);
                addtexturetoapp.setValue(text[0]);
                addtoshape_app.setValue(appear[0]);
                addtoshape_geom.setValue(geom[0]);
                addtotransf.set1Value(0,shape[0]);

             // Adding the nodes to the main node
                addtoanchor.set1Value(0,transf[0]);
                addtoancetre.set1Value(2,anch[0]);

} // end of addRow method

public void callback(EventOut value, double timestamp, Object data)
        {

        }
Color getColor (String col)
{
  // from a string (representing a color), it returns a Color objects.
  // ...
}
```

```
String buildDescription (String xcol, String xvalue,
                         String ycol, String yvalue,
                         String zcol, String zvalue,
                         String sizecol, String sizevalue,
                         String transcol, String transvalue,
                         String colorcol, String colorvalue,
                         String shapecol, String shapevalue,
                         String textcol, String textvalue)
  {
    //building of the description string from these different parameters
    return description;
  } // end of buildDescription

}//end of the ScatterChart class
```

III. The Datum Class

```
import java.util.*;

public class Datum extends Object
{

 Vector values; // values (strings) used to form the chart

 /*
    The general constructor that will be used to instantiate
    this class with all the necessary values
 */
 public Datum(Vector currentRow) {
    set_values (currentRow);
 }

 /**
    The set methods
 */
        public void set_values(Vector val)
        {
                values = val;
        }

        public Vector get_values()
        {
                return values;
        }

        public int get_values_size()
        {
                return values.size();
        }

        public String get_values_at (int i)
        {
                if (i <= values.size()) {
                        String temp = values.elementAt(i).toString();
                        return temp;
                }
                else {
                        return "";
                }
        }
}//end of the Datum class
```

IV. The Representation Class

```
//this class describes how we want to visualize a specific column

public class Representation extends Object {

        String dimension; // The dimension value (colour, shape, etc). But for the
                BarChart, only the three axes.

        String datavalues []; // The different values for a column.

        String dimvalues []; // The different values for the dimension specified in the
                dimension field.


        public Representation (String columnDimension,
                                String columnDatavalues [],
                                String columnDimvalues []) {
                dimension = columnDimension;
                datavalues = new String [columnDatavalues.length];
                datavalues = columnDatavalues;
                dimvalues = new String [columnDimvalues.length];
                dimvalues = columnDimvalues;
        }

        public String get_dimension()
        {
                return dimension;
        }

        public String get_datavalues_at (int i)
        {
                if (i < datavalues.length) {
                        return datavalues[i];
                }
                else {
                        return "";
                }
        }

        public String get_dimvalues_at (int i) {
                if (i < dimvalues.length) {
                        return dimvalues[i];
                }
                else {
                        return "";
                }
        }
}//end of the Representation class
```

V. *The Semantic Class*

```
import java.util.*;

//This class describes the semantic of a column
public class Semantic extends Object
{
 String name;// The name of the column (e.g. Gender).

 String type;//The type of the column (for Gender we will have the 'nominal' type).

 String legend;//The legend one wants to give for the association.

 Representation rep;//This associates a dimension with the column. See the
        Representation class

 public Semantic (String columnName, String columnType,
              String columnLegend, Representation columnRep)   {
    name = columnName;
    type = columnType;
    legend = columnLegend;
    rep = columnRep;
 }

 public String get_name() {
     return name;
 }

 public String get_type() {
     return type;
 }

 public String get_legend() {
    return legend;
 }

 public Representation get_rep() {
    return rep;
 }
}}//end of the Semantic class
```